Négociation de contrats dans les systèmes à composants logiciels hiérarchiques

Hervé Chang

▶ **To cite this version:**

Hervé Chang. Négociation de contrats dans les systèmes à composants logiciels hiérarchiques. Génie logiciel [cs.SE]. Université Nice Sophia Antipolis, 2007. Français. NNT : 2007NICE4069 . tel-00782493

HAL Id: tel-00782493

https://theses.hal.science/tel-00782493

Submitted on 30 Jan 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

École doctorale STIC
Sciences et Technologies de l'Information et de la Communication

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention : Informatique

présentée et soutenue par

Hervé CHANG

Négociation de contrats dans les systèmes à composants logiciels hiérarchiques

---

Thèse dirigée par Philippe COLLET et Philippe LAHIRE

soutenue le 06 décembre 2007, à Polytech'Nice Sophia-Antipolis

Jury :

M. Michel RUEHER	Professeur, UNSA, I3S	Président
Mme Françoise ANDRÉ	Professeur, Université de Rennes I, INRIA	Rapporteur
Mme Laurence DUCHIEN	Professeur, Université de Lille I, LIFL	Rapporteur
M. Thierry COUPAYE	Chercheur industriel expert, France Télécom R&D	Examinateur
M. Philippe COLLET	Maître de conférences, UNSA, I3S	Co-directeur de thèse
M. Philippe LAHIRE	Professeur, UNSA, I3S	Directeur de thèse

Remerciements

Je tiens à remercier ici chaleureusement les nombreuses personnes qui m'ont permis de réaliser cette thèse.

En tout premier lieu, je remercie les membres du jury pour avoir accepté d'évaluer ce travail de thèse et de participer au jury de soutenance :

- Mesdames Françoise André et Laurence Duchien pour avoir accepté la charge de rapporter ce mémoire ainsi que pour les nombreux retours et remarques sur ce document et ma soutenance ;
- M. Thierry Coupaye pour avoir bien voulu participer au jury et examiner mon travail ;
- M. Michel Rueher pour m'avoir fait l'honneur de présider le jury ;
- et MM. Philippe Collet et Philippe Lahire pour avoir dirigé ma thèse.

Je remercie chaleureusement Philippe Collet pour avoir dirigé mes recherches en stage de DEA et pendant ces trois années de thèse. Merci d'avoir dirigé mes travaux avec beaucoup de soin et d'attention, pour ta disponibilité sans faille ainsi que les retours nombreux et minutieux. Merci aussi de la confiance et de la sympathie que tu m'as témoignées ainsi que pour les nombreuses discussions autour de la thèse. J'ai beaucoup appris et apprécié travailler à tes cotés.

Je remercie également Philippe Lahire pour les relectures de ce mémoire, de même que Roger Rousseau pour les discussions informelles sur mon travail et leurs conseils avisés.

Enfin, je remercie les chefs d'équipe, Roger Rousseau, Philippe Lahire, et Michel Riveill, pour m'avoir accueilli dans les équipes OCL et Rainbow et permis de réaliser cette thèse dans les meilleures conditions, ainsi que tous les membres de l'équipe pour les encouragements et les diverses discussions.

Dans ce chemin qui ne devait pas forcément m'amener à la thèse, je souhaite aussi remercier, Patrick Capolsini et Jean-Marie Goursaud à l'Université de Polynésie-Française, Jean-Clarence Nosmas, Gilles Menez, Anne-Marie Pinna-Dery à l'Université de Nice Sophia-Antipolis, qui ont été à des moments opportuns à l'écoute de mes préoccupations et ont apporté un éclairage particulier dans mes réflexions.

J'ai bien évidemment une pensée pour tous mes amis, y compris ceux à Tahiti qui doivent certainement se demander où je suis et ce que je fais...Je ne vous ai pas oublié !

Enfin, je tiens à remercier toute ma famille : Papi, Mamie, Hiro, Catherine & Christine pour avoir accepté le fait que je sois si loin de vous pendant ces années, de même que mes retours effectués à dose homéopathique. Merci pour votre amour, votre soutien sans faille, et pour avoir toujours accepté mes choix sans les rendre plus compliqués que ce qu'ils pouvaient l'être.

Merci aussi à ma belle-famille, Jean-Claude, Gisèle et Fabrice pour m'avoir accueilli chez les 'KZA'. J'apprécie très sincèrement l'attention que vous me portez continuellement depuis maintenant quelques années.

Mes derniers mots vont à Virginie pour avoir été à mes cotés au quotidien pendant ces années. Merci d'avoir consenti à me partager, de façon certainement inéquitable, avec cette thèse, et de ton infinie patience à chaque moment. Merci enfin de ton soutien inconditionnel qui m'est d'une grande valeur.

Résumé

Les systèmes logiciels modernes sont caractérisés par leur complexité croissante et les fortes exigences en termes de continuité et de disponibilité des services. Face à cette problématique, des propositions telles que le génie logiciel à composants et l'approche contractuelle constituent des approches pertinentes pour faciliter la construction des systèmes et augmenter leur fiabilité. En outre, les aspects extrafonctionnels des systèmes à l'exécution doivent être constamment mieux gérés, et des techniques qui permettent aux systèmes logiciels de réagir à divers changements et maintenir des qualités satisfaisantes sont requises.

La contribution de cette thèse est ainsi un modèle d'auto-adaptation dynamique pour des systèmes logiciels fondés sur ces deux approches. Notre proposition originale consiste à définir un modèle de négociation automatisé de contrats qui permet aux composants contractualisés de conduire eux-mêmes l'auto-adaptation. Le modèle identifie les différents éléments de base permettant aux composants d'effectuer l'adaptation des composants ou des contrats, et chaque processus de négociation exploite diverses informations fines contenues dans les contrats. Des politiques de négociation différentes sont aussi développées pour orienter et fournir plusieurs possibilités de déroulement de négociation. En particulier, un support compositionnel permet de conduire la négociation dans les hiérarchies de composants. Le modèle de négociation s'appuie sur les principes généraux de la plate-forme à composants hiérarchiques *Fractal* et du modèle de contrats *ConFract*, et son intégration est effectuée sur ces deux technologies. Les propositions sont illustrées sur des exemples extraits d'un cas d'étude et les premières expérimentations du modèle sont présentées.

Mots-clés: Génie logiciel des composants, Composants logiciels hiérarchiques, Contrats, Aspects extrafonctionnels, Négociation dynamique, Fractal, ConFract

Abstract

Modern software systems are characterized by their increasing complexity and the strong requirements with respect to the continuity and availability of services. To address these needs, the component-based software engineering and the contractual approach represent two relevant approaches which aims at facilitating the construction of larger systems and increasing their reliability. In addition, non-functional properties of running systems have still to be better managed and some mechanisms that allow them to react to changes and to maintain some satisfactory quality levels are required.

This thesis proposes a self-adaptation model for runtime systems that are based on both software components and contracts. Our proposal consists in defining an automated contract negotiation model which makes components negotiate themselves the contract in which their are involved. The negotiation model identifies the different basis elements required to make components operate the adaptation of the components or the contracts, and each negotiation process takes advantage of the various information contained in the contracts. Two negotiation policies are also designed to conduct the negotiation process in different ways. In particular, a compositional reasoning support is proposed to conduct the negotiation processes down into component hierarchies. The negotiation model is based on the general principles of the hierarchical software components platform, named *Fractal*, and on the contract model, named *ConFract*, and it is integrated into these two technologies. The contributions are illustrated on various examples which are extracted from a larger case study and the first experiments are described.

Keywords: Component-based software engineering, Hierarchical software components, Contracts, Non-functional aspects, Dynamic negotiation, Fractal, ConFract

Table des matières

1 Introduction

Le logiciel est devenu, de nos jours, un élément central et omniprésent dans des domaines aussi variés que l'économie, l'industrie, le domaine médical, l'automobile ou encore l'administration. Il ne représente plus seulement la part technique et marginale des systèmes, mais est devenu le cœur de fonctionnement et l'intelligence qui contrôle les systèmes modernes. Ainsi, les systèmes logiciels actuels réalisent des *services* très divers et ont acquis une importance telle que les *fonctionnalités* qu'ils offrent constituent leur valeur ajoutée et représentent leurs caractéristiques majeurs et discriminantes. Compte tenu de cette utilisation répandue et intensive, les contraintes sur les systèmes logiciels sont de plus en plus fortes. Ils doivent offrir de plus en plus vite des fonctionnalités de plus en plus riches ; celles-ci doivent satisfaire à des exigences en terme de *qualité* ; les systèmes ont aussi une durée de vie plus longue et ils doivent donc régulièrement évoluer pour répondre à de nouveaux besoins et convenir à des contextes d'utilisation de plus en plus variés (distribution, hétérogénéité, etc.). Toutes ces contraintes ont pour conséquence directe d'augmenter radicalement la taille et la complexité des systèmes logiciels, et d'introduire une pression supplémentaire sur les délais ainsi que sur les coûts de développement et de maintenance.

Pour maîtriser la construction et l'évolution des systèmes logiciels, le génie logiciel propose continuellement divers principes, méthodes et techniques ayant pour objectif final de produire, dans une approche systématique, du logiciel de *qualité* élaboré en respectant les contraintes de délais et de coûts et qui satisfasse aux besoins des utilisateurs finaux. Pour répondre à cet objectif, le développement logiciel doit, en particulier, faire face à deux défis majeurs que sont l'*augmentation croissante de la complexité* des systèmes et le besoin croissant de les *adapter à leur environnement ainsi qu'à de nouveaux besoins*. Les techniques de génie logiciel sont donc contraintes à évoluer sans cesse pour permettre de maîtriser cette complexité et d'offrir une certaine capacité d'adaptation. Cette évolution se fait notamment en introduisant de nouvelles abstractions à divers niveaux (langages, plates-formes, architectures, etc.), tout en continuant à s'appuyer sur des notions fortes déjà existantes telles que la réutilisation [McIlroy 1968] et la modularisation [Parnas 1972].

Dans cette perspective, le *génie logiciel à composants* [Heineman et Councill 2001; Szyperski 2002] a récemment été mis en avant comme étant une approche pertinente qui applique ces principes et promet *a priori* une gestion plus efficace de la complexité, une réduction significative des temps/coûts de développement ainsi qu'une productivité accrue [Brown 2000]. Dans cette approche, les *composants* encapsulent des fonctionnalités bien définies, et exposent clairement des interfaces fournies et requises qui

décrivent les fonctionnalités qu'ils fournissent et requièrent. Ils représentent alors des briques fonctionnellement unitaires aux dépendances explicites et fortement réutilisables. Ces composants préalablement développés peuvent alors être assemblés pour construire, de façon incrémentale, des systèmes logiciels plus larges. De plus, de nouvelles fonctionnalités sont apportées ou retirées des systèmes en les *reconfigurant*, par exemple, par des ajouts ou retraits de composants, et des modifications de leurs connexions.

Comme l'approche par composants contribue à rendre plus explicite la structure des systèmes logiciels au travers des composants, elle entretient des liens très étroits et complémentaires avec l'*architecture des systèmes* [Bass *et al.* 1998]. Cette dernière discipline se concentre, à un niveau général, sur la description de l'organisation et de la structure des systèmes (unités, propriétés visibles, relations, etc.) et intervient surtout dans les premières phases de développement pour évaluer la satisfaction aux exigences. L'approche par composants s'efforce de prolonger et de maintenir l'architecture des systèmes aussi lors des phases de déploiement et d'exécution. Ainsi, dans les systèmes à composants avancés, les composants et la structure des systèmes restent explicites aussi à l'exécution, et l'approche par composants maintient donc un continuum d'architecture entre les différentes phases du cycle de vie des systèmes (développement, déploiement, configuration, exécution, reconfiguration). De plus, ils autorisent aussi maintenant les compositions hiérarchiques de composants (des composants peuvent être assemblés récursivement pour former un autre composant) [Bruneton *et al.* 2006], et offrent aussi des capacités réflexives (introspection et intercession) de plus en plus étendues [Kon *et al.* 2000; Coulson *et al.* 2002; Bruneton *et al.* 2006].

L'approche à composants est appliquée de façon répandue et avec succès à la fois dans les milieux académique et industriel, et de nombreuses technologies à composants, se situant à différents niveaux d'abstraction et couvrant un large spectre de classes d'applications (systèmes d'exploitation, systèmes embarqués, intergiciels, etc.) ont été proposées pour soutenir le développement par composants. Toutefois, le génie logiciel à composants n'a pas encore complètement réussi. Les attentes restent fortes et de nombreux défis doivent encore être résolus. En particulier, alors que les aspects fonctionnels des composants et des systèmes sont par nature plutôt clairement spécifiés au travers des interfaces fournies et requises, tous les aspects autour des caractéristiques *extrafonctionnelles* des composants ne sont pas encore complètement résolus, et un verrou majeur réside notamment dans le manque de support pour gérer les aspects extrafonctionnels des composants. Ces aspects extrafonctionnels ne concernent pas directement les services fournis et requis des composants mais décrivent plutôt d'une façon très générale les *qualités* des systèmes et les conditions dans lesquelles les fonctionnalités sont rendues. En outre, comme ils sont définis simplement par opposition aux aspects fonctionnels, les aspects extrafonctionnels sont réellement divers, et un premier problème apparaît donc dans l'existence de définitions à la fois nombreuses et souvent imprécises. Dans la pratique, ces aspects sont considérés de façon très variée ; ils peuvent être abordés selon différents points de vue, et différents degrés d'importance peuvent aussi leur être accordés. Par exemple, du point de vue d'utilisateurs finaux, des exemples de propriétés extrafonctionnelles importantes sont représentées par des qualités générales des systèmes telles que la simplicité d'installation, la facilité d'utilisation ou encore un *bon* comportement attendu. D'autres propriétés extrafonctionnelles cette fois-ci pertinentes du point de vue du fonctionnement des systèmes, décrivent par exemple la robustesse, la consommation en ressources ou encore la qualité des services rendus [Barbacci *et al.* 1995]. En conséquence, en plus de la réalisation stricte des services, les propriétés extrafonctionnelles des systèmes logiciels actuels sont devenues de première importance et constituent un critère de différenciation à fonctionnalités équivalentes.

1.1 Problématiques majeures

Comme discuté précédemment, tout système logiciel à partir d'une certaine taille fait nécessairement intervenir diverses propriétés extrafonctionnelles. En même temps que les fonctionnalités des systèmes s'enrichissent, les aspects extrafonctionnels des systèmes doivent être mieux couverts et pris en compte afin de toujours satisfaire les besoins des utilisateurs et les qualités de fonctionnement des systèmes. Par ailleurs, comme les propriétés extrafonctionnelles sont très diverses, couvrent tout le cycle de vie des systèmes et que ce dernier tend justement à s'allonger, les propriétés extrafonctionnelles doivent aussi être mises en relation selon ce même cycle de vie. En particulier, elles doivent être prises en compte le plus tôt possible ; elles doivent évoluer avec les besoins, les utilisations et les contextes d'exploitation des systèmes, et elles doivent être gérées de différentes manières en fonction des phases du cycle de vie (déploiement, configuration, exécution, etc.). Dans ce contexte, deux problématiques liées à la gestion des propriétés extrafonctionnelles se posent. Il est important de pouvoir *définir et identifier les propriétés extrafonctionnelles* mises en jeu. Lorsque ces dernières sont mal ou très faiblement connues, il devient très difficile d'avoir une idée précise des comportements généraux des systèmes et des niveaux de qualités résultants, et cela a pour conséquence de limiter le niveau de confiance que l'on peut leur accorder (mauvaises utilisations, erreurs, comportements inattendus,...). De plus, comme de nombreuses propriétés extrafonctionnelles sont relatives au comportement des systèmes à l'exécution et qu'elles *varient* lors de celle-ci, il est primordial de pouvoir *maintenir des niveaux de qualités acceptables* malgré les fluctuations dans le fonctionnement ou l'environnement des systèmes. De telles fluctuations sont par exemple des variations de ressources (mémoire, processeur, etc.), des variations d'environnement (environnements de déploiement, plates-formes hétérogènes, etc.), des changements de mode de fonctionnement, etc. La gestion des propriétés extrafonctionnelles *à l'exécution* est importante de fait puisque ces propriétés ont un impact particulier sur la qualité des services rendus par le système et visibles à l'exécution par les utilisateurs finaux. De plus, comme la tendance actuelle est aussi d'offrir des systèmes qui fonctionnent *en continu* et doivent maintenir des services *hautement disponibles*, cette phase d'exécution des systèmes acquiert une place encore plus importante dans son cycle de vie.

Pour maintenir des niveaux de qualités attendus, des approches classiques, surtout issus des travaux dans le domaine de la construction fiable de systèmes ou des télécommunications et réseaux, proposent de garantir un certain niveau de qualité par construction des systèmes ou par dimensionnement des ressources de l'environnement d'exécution. Parallèlement à ces approches, et devant le constat que toutes les propriétés ne peuvent pas être complètement modélisées et garanties avant l'exécution des systèmes, de nombreux axes de recherche se sont aussi développés autour de l'*adaptation* des systèmes lors de leur exécution. L'*adaptation dynamique* représente la capacité de modifier le comportement de sous-parties d'un système lors de son exécution. Par exemple, cela peut-être motivé par le besoin de modifier une partie d'un système pour introduire de nouvelles fonctionnalités, ou pour ce qui concerne plus notre cadre d'étude, par la nécessité de faire évoluer et reconfigurer des systèmes qui fonctionnent en continu en réponse à de nouveaux besoins (utilisateurs, applications, etc.) ou des changements d'environnement (ressources, niveaux de services, etc.). Les systèmes adaptatifs se déclinent entre autres en deux sous-classes selon qu'ils soient *adaptables* ou *auto-adaptatifs*. Les systèmes adaptables fournissent des moyens qui permettent uniquement à des acteurs externes de les adapter explicitement [Blair *et al.* 2001], mais ces adaptations peuvent être opérées potentiellement à n'importe quel moment y compris à l'exécution. A l'inverse, les systèmes auto-adaptatifs ne requièrent pas d'intervention extérieure [Oreizy *et al.* 1999]. Ils fonctionnent en boucle fermée et intègrent des logiques d'adaptation, souvent définies et intégrées dès la construction des systèmes, qui leur permettent de *surveiller activement* des conditions de leur fonctionnement et de *réagir eux-mêmes* à des changements de leur état ou de l'environnement. Ainsi, ces systèmes auto-adaptatifs possèdent des logiques qui décrivent *quand* et *comment* réagir aux événements bien précis, et en fournissant des moyens pour modifier les systèmes de façon autonome lors

de leur exécution, ces technologies d'auto-adaptation à l'exécution constituent aussi une base fondatrice pour l'élaboration de qualités d'auto-régulation plus générales (surveillance, configuration, réparation, optimisation, protection) dans le contexte émergent des systèmes autonomiques [Kephart 2005; Parashar et Hariri 2005].

Le sujet de cette thèse s'inscrit dans ce cadre de l'auto-adaptation dynamique des systèmes construits par assemblages de composants logiciels. Actuellement, alors que les systèmes sont construits par intégration de composants dans une approche montante (*bottom-up*), la plupart des technologies d'adaptation pour construire des systèmes auto-adaptatifs sont encore réalisées dans une approche descendante (*top-down*). Ainsi, elles s'appuient sur des techniques, infrastructures ou modèles de performances qui prennent en charge l'auto-adaptation sous une connaissance globale des systèmes, sans s'appuyer sur les notions de décomposition et d'architecture explicite, y compris à l'exécution, apportées par l'approche à composants. De plus, la surveillance et la réalisation des adaptations sont aussi souvent gérées par une entité centralisée gestionnaire d'adaptation au niveau d'une infrastructure sous-jacente, alors que les logiques d'adaptations pourraient être réparties au niveau des composants et les adaptations réalisées par des interactions entre composants.

Ayant reconnu primordiales les problématiques soulevées et les points faibles des approches traditionnelles, une question naturelle se pose pour savoir « *s'il est possible d'organiser différemment l'auto-adaptation des systèmes à composants, cette fois-ci, en s'appuyant davantage sur le fait que les composants représentent les unités de décomposition des systèmes, pour structurer et répartir la réalisation de l'auto-adaptation du système au niveau de ses composants* ».

Ainsi, à la différence des mécanismes traditionnels qui opèrent de façon centralisée et requièrent une connaissance globale, l'approche générale proposée dans cette thèse consiste à prolonger le rôle d'unité de décomposition et de fonctionnement qu'ont les composants, pour leur confier la *responsabilité de conduire eux-mêmes* l'adaptation dynamique de leur système. Les composants représentent déjà les briques fonctionnelles de base et interagissent pour réaliser les services offerts par leur système. A l'exécution, le système peut être soumis à des variations de fonctionnement et de l'environnement. Dès lors, pour réagir à ces variations, l'auto-adaptation est répartie au niveau des composants. Ils prennent en charge et implémentent l'auto-adaptation de leur système en interagissant entre eux.

1.2 Démarche

Pour mettre en œuvre de tels mécanismes d'auto-adaptation, il est nécessaire de pouvoir i) effectuer des observations de façon à détecter les variations qui surviennent dans le fonctionnement des systèmes ou dans leur environnement, et ii) définir les mécanismes qui vont faire interagir les composants pour qu'ils conduisent eux-mêmes l'adaptation. Pour répondre à ces besoins, nos travaux reposent sur deux approches. Ils s'appuient fortement sur une application de l'approche contractuelle (*Design By Contract*) [Meyer 1992a] à des composants logiciels. De plus, ils s'inspirent de certains mécanismes d'interactions proposés dans le domaine des systèmes multi-agents.

Le principal objectif de l'approche contractuelle est de fournir au travers de la notion de *contrat logiciel*, une base explicite et efficace pour spécifier et vérifier le plus précisément possible les collaborations entre les différents éléments logiciels, en terme d'obligations et de bénéfices respectifs. Initialement introduite pour la programmation par objets, les contrats spécifient usuellement dans ce cadre des pré- ou post-conditions sur les méthodes ou encore des invariants de classes. Appliqués aux composants logiciels, ces contrats sont des spécifications d'obligations qui doivent aussi prendre en compte les diverses caractéristiques des composants (interfaces fournies et requises, assemblages, composants, etc.) et couvrir davantage le spectre des éléments fonctionnels et extrafonctionnels [Beugnard *et al.* 1999]. De plus, ils doivent aussi *suivre le cycle de vie* des systèmes et, contenir et maintenir à jour diverses informations

sur le contexte de spécification et de vérification (contraintes, propriétés et éléments spécifiés, contexte d'évaluation, etc.). De telles informations doivent aussi être rendues accessibles à l'exécution, puisque l'adaptation opère à ce moment. De cette façon, comme les contrats effectuent des vérifications sur diverses propriétés et éléments des composants, ils permettent de réaliser des observations sur diverses sous-parties du système et de localiser des variations ou erreurs liées au fonctionnement ou à l'environnement. Lors des fluctuations des propriétés extrafonctionnelles, les cas d'erreurs ou de fautes sont détectées par les contrats et entraînent leurs *violations*. Ces violations de contrats sont alors très fréquemment gérées par une politique restrictive qui consiste à rejeter l'opération et déclencher des signaux d'erreurs en attribuant un blâme à la partie responsable de la faute. Toutefois, il est aussi potentiellement possible d'envisager d'autres types de traitements tels que i) ignorer la violation du contrat comme si ce dernier était absent ou désactivé, ii) attendre que d'autres potentielles opérations changent l'état du contrat violé, ou encore iii) négocier les termes du contrats en faisant intervenir les parties concernées. Par ailleurs, comme les contrats formalisent justement les obligations entre les différents composants, les différentes parties impliquées dans un contrat sont clairement connues et diverses formes de *responsabilité* peuvent en plus leur être attribuées, compte tenu des différents rôles qu'elles ont dans les collaborations. Ces responsabilités permettent alors de comprendre plus finement les erreurs détectées et, outre l'attribution de blâmes aux parties fautives, elles peuvent être exploitées pour élaborer des mécanismes d'auto-adaptation visant à rétablir les contrats et adapter dynamiquement le système.

Dans notre approche de l'auto-adaptation, comme la spécification et la vérification des propriétés sont organisées autour des contrats et que ces derniers contiennent diverses informations riches et à jour sur lesquelles il est possible de raisonner, nous soutenons l'idée selon laquelle les mécanismes de rattrapage doivent aussi faire partie de la contractualisation, en plus de s'appuyer sur la structuration en composants motivée précédemment. Par conséquent, nous prolongeons l'approche contractuelle en organisant l'auto-adaptation par des *mécanismes de négociation automatisés de contrats entre composants* dans le but de rétablir la validité des contrats. Ainsi, de la même façon que les contrats sont définis pour régir le fonctionnement des composants, la négociation de contrats trouve naturellement sa place en permettant à ces derniers de conduire et négocier eux-mêmes le contrat violé dans lequel ils sont impliqués de sorte à adapter le système et à rétablir le contrat. Le rétablissement du contrat en sortie de l'auto-adaptation permet alors de s'assurer du retour à un état valide de la sous-partie du système impactée.

Dans son sens le plus général, la négociation constitue « *un processus par lequel deux ou plusieurs parties échangent des propositions pour parvenir à un un accord mutuel* » [Pruitt 1981]. Elle représente donc une notion fondamentale qui permet de façon générale de gérer dynamiquement les interactions d'*entités* dans un contexte où ils ont individuellement une certaine intelligence, et doivent collectivement s'organiser de façon distribuée dans un environnement qu'ils partagent (organisation d'un travail commun, planification cohérente d'actions, etc.). En revanche, la notion de négociation est très générale et reste souvent difficile à appréhender en raison des nombreux concepts qu'elle intègre mais aussi de son applicabilité dans divers contextes. En particulier, elle est couramment utilisée dans le domaine des systèmes multi-agents en tant que mécanisme puissant pour supporter la coopération ou la coordination d'agents logiciels qui ont des dépendances et sont fortement *autonomes* [Briot et Demazeau 2001].

Vis-à-vis de la négociation de contrats logiciels, il apparaît naturel que les composants qui sont liées par les contrats soient assimilables aux participants d'une négociation, si un contrat est violé. Pour concevoir notre modèle de négociation, l'étude de la négociation dans les systèmes multi-agents sera donc effectuée en analysant les mécanismes sous-jacents et les différentes composantes nécessaires, avec une attention particulière sur le protocole de négociation, les objets négociés, et le raisonnement des composants. Compte tenu de notre objectif, le modèle de négociation proposé sera aussi élaboré dans une approche pragmatique qui prend en compte les spécificités de l'approche par composants, celles des contrats logiciels ainsi que les contraintes de l'auto-adaptation des systèmes. Les diverses informations contenues dans les contrats seront exploitées et des modélisations explicites des propriétés extrafonc-

tionnelles compositionnelles seront aussi utilisées pour piloter la négociation dans les hiérarchies de composants et offrir un minimum de contrôle dans le déroulement de celui-ci. De plus, des contrats négociables seront aussi définis sur le système de négociation lui-même, de façon à contrôler voire adapter les processus de négociation. Par ailleurs, les mécanismes d'auto-adaptation proposés dans le cadre de notre modèle de négociation ainsi que l'autonomie résultante des systèmes seront aussi évalués dans la perspective connexe de l'approche autonomique qui vise à créer des systèmes informatiques capables de s'autogérer.

1.3 Hypothèses principales

L'objectif de cette thèse est de concevoir un modèle de négociation pour composants logiciels contractualisés. Cette section présente les principales hypothèses de nos travaux.

Génie logiciel à composants. Nous prenons comme point de départ de nos travaux l'approche par composants et les systèmes construits à partir de composants logiciels. En particulier, nos travaux s'appuient sur la plate-forme de composants logiciels *Fractal* [Bruneton *et al.* 2006]. Ce modèle de composants récemment introduit est un canevas de conception, général et modulaire qui offre différentes qualités intéressantes en plus de reprendre les principales caractéristiques des modèles de composants existants (encapsulation, séparation entre interfaces et implémentations, séparation des préoccupations, description explicite des dépendances, description architecturale, etc.). Ainsi, il supporte l'organisation hiérarchique et le partage de composants, maintien la notion de composant aussi à l'exécution, offre des capacités réflexives et des possibilités de reconfiguration dynamique, et effectue la gestion des aspects techniques des composants par des entités dédiées en suivant une approche par séparation des préoccupations. Le modèle présente aussi l'avantage de pouvoir servir de base à diverses extensions [Coupaye *et al.* 2008] et il propose une plate-forme complètement opérationnelle qui supporte différents langages et types d'implémentation. Toutes ces caractéristiques en font donc un support idéal pour nos recherches.

Approche contractuelle. Pour effectuer la spécification et la vérification des propriétés, nos travaux s'appuient sur une application de l'approche contractuelle aux composants logiciels. Dans cette optique, un système de contractualisation, nommé *ConFract* [Collet *et al.* 2005] et précédemment défini dans l'équipe RAINBOW du Laboratoire I3S, applique l'approche contractuelle à des composants hiérarchiques tels que ceux définis dans le modèle de composants *Fractal*. Ce système de contrats définit des caractéristiques intéressantes en adéquation avec notre démarche. Les contrats sont des objets de première classe qui sont construits à partir de spécifications. Ils permettent de spécifier et vérifier diverses propriétés sur les différents éléments d'architecture des composants (interfaces, assemblages interne et externe), attribuent diverses responsabilités aux composants et suivent le cycle de vie des composants. Cette caractéristique est fondamentale pour effectuer des vérifications diverses lors de l'exécution du système, car les contrats peuvent ainsi observer le fonctionnement des composants et reporter des erreurs. Lorsqu'ils sont violés, la politique actuelle du système consiste basiquement à signaler les erreurs ainsi que les composants responsables concernés. Pour mettre en œuvre complètement l'auto-adaptation dans les systèmes à composants contractualisés, les mécanismes de négociation sont bâtis sur le système *ConFract*, et le travail de cette thèse s'inscrit donc en directe extension de ce système.

Approche par négociation Il existe plusieurs façons de mettre en œuvre l'auto-adaptation dans les systèmes. Dans le cadre des systèmes construits par assemblage hiérarchique de composants logiciels, la réponse à la question principale posée dans la problématique est qu'il nous semble possible de s'appuyer sur les composants pour répartir et conduire l'auto-adaptation. Ainsi, l'approche défendue dans cette thèse consiste à proposer de nouvelles techniques d'auto-adaptation dans les composants en se

basant sur le concept de *négociation*. Comme les contrats régissent le fonctionnement des composants, la responsabilité de l'auto-adaptation leur est confiée. Les composants impliqués vont alors interagir et vont négocier dynamiquement pour réaliser l'auto-adaptation du système. Pour cela, les composants sont munis de certaines capacités et interagissent entre eux par les mécanismes de négociation automatisés de contrats proposés, et sur la base de leur responsabilité dans les contrats violés.

1.4 Contributions

La principale contribution de ce travail de thèse est la conception d'un modèle de négociation automatisé de contrats pour des composants logiciels hiérarchiques. La négociation porte sur des contrats logiciels et celle-ci est notamment pertinente lorsque les contrats spécifient des propriétés extrafonctionnelles. Elle s'opère surtout à l'exécution des systèmes pour réagir aux fluctuations des propriétés extrafonctionnelles. Elle vise de façon automatisée à adapter dynamiquement le système et à rétablir la validité des contrats lorsqu'ils sont violés. Plus précisément, les principaux éléments de contributions sont les suivants :

- la définition des éléments de base nécessaires à la mise en œuvre de mécanismes de négociation pour composants logiciels contractualisés ;
- le développement de deux politiques différentes de négociation qui offrent clairement deux déroulements possibles des négociations et qui se distinguent du point de vue de leur portée et des éléments qu'elles adressent. La politique par *concession* exploite la responsabilité de bénéficiaires des contrats, vise à effectuer des adaptations des contrats et reste confinée au niveau de hiérarchie de la violation. La politique par *effort* s'appuie sur la responsabilité de garant des contrats, vise à effectuer des adaptations sur les composants et peut potentiellement se propager dans les hiérarchies de composants.
- le développement d'un support permettant d'intégrer des propriétés extrafonctionnelles dans les composants logiciels, et de raisonner sur les propriétés compositionnelles dans les hiérarchies des composants. Ce support permet d'expliciter certaines classes de propriétés extrafonctionnelles vis-à-vis des composants pris individuellement. De plus, il fournit des mécanismes permettant de raisonner simplement sur des propriétés compositionnelles des composants. Ce support est en particulier exploité dans la politique par effort pour propager les négociations dans les hiérarchies des composants ;

L'application de ces propositions est démontrée sur la plate-forme de composants *Fractal* et sur le système de contrats *ConFract*. Les propositions sont illustrées sur des exemples tirés d'une application de validation détaillée dans l'annexe A. Les contributions développées ainsi que leur utilisation sont aussi évaluées de façon qualitative et les résultats des premières expérimentations sont présentés.

1.5 Plan du document

La suite de ce document de thèse se décompose en neuf chapitres regroupés en deux parties.
La partie I présente l'état de l'art décrivant les trois principaux domaines de recherche connexes à ce travail.

Chapitre 2 Ce chapitre présente les principales approches contractuelles dans les composants logiciels. Il rappelle les principales approches contractuelles dans les objets et décrits les éléments spécifiés dans les contrats et les techniques de vérifications dans les composants. Ce chapitre présente aussi les contrats électroniques et les systèmes de gestion de contrats proposés dans ces environnements.

Chapitre 3 Ce chapitre présente une importante vision d'ensemble sur les aspects extrafonctionnels. Il présente rapidement les principales problématiques avant de décrire successivement dans une approche verticale, les principales classifications des aspects extrafonctionnels, les langages de spécifications de qualité et les mécanismes de gestion des aspects extrafonctionnels.

Chapitre 4 Ce chapitre présente un panorama complet des concepts et des travaux relatifs à la négociation dynamique. Il fournit une vision d'ensemble des concepts autour de la négociation dynamique en général et des ses approches alternatives. Puis, les principaux travaux issus du domaine des systèmes multi-agents sont présentés.

Chapitre 5 Ce chapitre conclut l'état de l'art en tirant des conclusions sur les travaux étudiés et en donnant le cahier des charges du modèle de négociation.

La partie II regroupe les contributions de ce travail de thèse.

Chapitre 6 Ce chapitre présente le modèle de négociation de contrats développé au cours de ce travail de thèse. Il commence par présenter les principes et les différents éléments de base du modèle. Puis, il détaille et illustrent les politiques de négociations conçues. Enfin, il effectue un retour sur les choix effectués.

Chapitre 7 Ce chapitre présente les patrons d'intégration des propriétés extrafonctionnelles et le support de raisonnement compositionnel associé aux propriétés compositionnelles. Les propositions sont d'abord présentées d'une façon abstraite avant d'être appliqué dans la plate-forme *Fractal*.

Chapitre 8 Ce chapitre décrit la mise en œuvre du modèle de négociation dans la plates-formes *Fractal* et *ConFract*. Cette mise en œuvre repose sur la version 2.5 de l'implémentation de référence *Julia* de la plate-forme *Fractal*, et sur la première version du système *ConFract*. Les principaux éléments d'architecture du système, leur implémentation et leurs interactions y sont présentés.

Chapitre 9 Ce chapitre regroupe les éléments d'évaluation du modèle de négociation. Une évaluation qualitative du modèle et des processus de négociation est tout d'abord présentée, puis quelques éléments méthodologiques sont décrits et enfin les résultats des premières évaluations de performances sont présentés.

Enfin, le chapitre 10 conclut ce document. Il effectue un résumé des contributions, présente les perspectives de recherches soulevées par ce travail de thèse et liste les publications liées à cette thèse.

1.6 Financement et contexte de travail

Ce travail de thèse a été financé par une allocation de recherche du ministère MENESR d'une durée de trois ans. Il a été réalisé au sein du projet OCL puis RAINBOW [1], du Laboratoire I3S [2] à Sophia Antipolis, dans une sous-thématique du projet qui s'intéresse à l'application des approches contractuelles dans les architectures orientées composants ou services. Les recherches ont aussi été menées en étroite collaboration avec le contrat de recherche externe I3S/France Télécom R&D n^o 46132097. Les résultats obtenus ont été publiés dans divers articles et rapports de recherche.

1. http ://rainbow.i3s.unice.fr
2. http ://www.i3s.unice.fr

PREMIÈRE PARTIE

ÉTAT DE L'ART

Buts et portée de la partie

L'objectif de cette première partie est de fournir un état de l'art des principaux travaux connexes à cette thèse. Les travaux présentés dans ce chapitre sont regroupés selon trois grands axes. Le premier chapitre effectue une brève étude des travaux autour des composants contractualisés et des systèmes de contractualisation. Le deuxième chapitre propose une vision d'ensemble sur les aspects extrafonctionnels, leurs définitions ainsi que les mécanismes de spécification et d'adaptation dynamique. Enfin, le troisième chapitre présente une étude des concepts et des mécanismes de négociation dynamique, en particulier issus du domaine des systèmes à agents.

2

Contrats dans les composants

La plupart des travaux qui portent sur les contrats dans les composants trouvent leurs origines dans les premières approches contractuelles dans les systèmes à objets. D'un point de vue strictement fonctionnel, dans les systèmes à objets ou à composants, des groupes d'entités coopèrent pour effectuer diverses tâches ou maintenir certains états du système. Les contrats visent alors à expliciter les fonctionnalités apportées par de telles entités ainsi que les interactions qui interviennent entre groupes d'objets ou de composants.

2.1 Contrats dans les objets

La notion de contrat telle qu'elle est actuellement acceptée est attribuée aux travaux de Meyer [Meyer 1992a]. Ces travaux se focalisent surtout sur le point de vue de l'interaction client-serveur entre deux objets, et ils sont menés dans l'objectif principal de réduire la complexité des systèmes à objets et d'augmenter leur fiabilité. De façon parallèle, une autre contribution importante est issue des travaux de Helm et al. [Helm *et al.* 1990] qui s'attachent davantage à expliciter les interactions entre objets coopérants dans une optique de réutilisabilité.

Contrats comportementaux

Les contrats définis par Helm et al. [Helm *et al.* 1990; Holland 1992] partent du constat selon lequel les interactions entre les objets représentent l'aspect fondamental et de première importance des systèmes à objets [Holland 2003]. Ces contrats ont donc été introduits dans l'objectif de rendre plus explicite les interactions entre les objets coopérants. De ce fait, ils visent donc à faciliter la conception et la réutilisabilité des groupes d'objets au delà de celles des simples classes. Chaque objet coopérant est vu comme étant un *participant* potentiel d'un contrat et ce dernier définit des obligations contractuelles associées à un ensemble de participants. Ces obligations se distinguent selon les *obligations de type*, qui portent sur le type des variables et des méthodes et les interfaces externes supportées par les objets), et les *obligations causales* qui capturent les dépendances comportementales entre objets, en terme de séquencement des actions ainsi que de satisfaction de préconditions, à l'instanciation du contrat, et d'invariants, pendant la durée du contrat. Pour pouvoir réutiliser des contrats existants et créer des contrats plus complexes,

ceux-ci supportent aussi la spécialisation (rajout ou surcharge des obligations) et l'inclusion (héritage et réunion des obligations).

Design By Contract

Avec son approche contractuelle « *Design By Contract* » (*DbC*) [Meyer 1992a], Meyer introduit les contrats ainsi qu'un ensemble de principes et de méthodologies qui visent à être appliqués dans une approche systématique pour faire face aux erreurs logicielles [Jézéquel et Meyer 1997] et augmenter la fiabilité des systèmes à objets. Les contrats logiciels de Meyer sont inspirés des contrats humains bilatéraux, et établis dans le cadre des interactions elles-mêmes bilatérales entre un client et un serveur. Ils explicitent les bénéfices et les obligations mutuels des parties qui s'expriment alors en terme de préconditions et de postconditions attachées aux opérations. Les préconditions expriment des contraintes devant être satisfaites par l'appelant d'une opération pour que celle-ci puisse se dérouler correctement, alors que les postconditions expriment des contraintes qui sont satisfaites par l'opération en sortie de celle-ci. De plus, un autre élément important défini dans *DbC* est la notion d'invariant de classe qui représente une propriété qui s'applique et doit être satisfaite pour toutes les instances d'une classe. Les contrats de Meyer font partie intégrante du langage *Eiffel* [Meyer 1992b] et sont aussi appliqués de façon par extension à d'autres langages de programmation, comme *Java* [Findler *et al.* 2001]. Les pré et postconditions sont implémentées par des assertions formées par des expressions booléennes. Elles sont vérifiées à l'exécution des programmes, et les contrats non satisfaits entraînent alors des levées d'exception qui indiquent des erreurs, soit vis-à-vis de l'appelant pour une précondition, soit de l'appelé pour une postcondition. Dans le cas général et compte tenu de leur expresssivité, les spécifications par des assertions ne sont pas décidables pour toute exécution, et donc non vérifiables statiquement. Les faiblesses des assertions comparées à d'autres techniques de spécification plus formelles sont reconnues. Toutefois, la spécification par des assertions constitue un bon compromis pour mettre en œuvre facilement des techniques de fiabilité dans les systèmes logiciels [Meyer 1992a]. Ainsi, l'approche *DbC* présente de nombreux avantages (documentation automatique, aide à la conception, exploitation pour les tests) et elle est devenue populaire car elle permet, non seulement de spécifier les bénéfices et les obligations des opérations, mais fournit aussi, dans une approche systématique, des mécanismes intégrés aux langages pour vérifier les contrats objets, client-serveur, et signaler leurs violations.

2.2 Contrats dans les composants

Cette section présente les éléments essentiels des principales propositions de contrats dans les composants et apporte une conclusion comparative.

Spécifications d'interface

Büchi et Weck [Buchi et Weck 1997] définissent un langage conçu comme une extension de Java pour définir des spécifications sur les interfaces des composants. Les contrats ressemblent très fortement aux travaux initiaux de Helm et Holland dans les objets, si ce n'est qu'ils sont directement intégrés au langage Java et que les spécifications peuvent référer des éléments privés des composants, lesquels doivent alors être vus comme des boîtes grises [Buchi et Weck. 1999].

Han [Han 1998, 2000] présente un canevas qui vise à spécifier l'utilisation des interfaces de composants. Le canevas veut prendre en compte les aspects des interfaces relatifs à leur signature (syntaxe des attributs et des opérations accessibles), leur configuration (ports fournis et requis), leur comportement (pré et postconditions), leur interaction (protocole d'utilisation) et leur qualité (mentionnée sans plus de détails). Han met en avant les qualités de simplicité (notations proches des IDLs) et de légèreté.

Niveaux de contrats

Beugnard et al. [Beugnard *et al.* 1999] définissent une classification de quatre niveaux de contrats dans les composants. Les contrats *basiques* décrivent les éléments syntaxiques des opérations des composants. Les contrats *comportementaux* spécifient le comportement des opérations en termes de pré et postconditions et d'invariants de classe. Les contrats de *synchronisation* spécifient le comportement global des composants en termes de synchronisation des diverses invocations des opérations, et permet, par exemple, de décrire les relations de séquence et de parallélisme. Enfin, les contrats de *qualité de service* expriment les paramètres de qualité requis et fournis. Ces contrats couvrent différents aspects relatifs aux opérations des composants. Ils sont mis en relation avec le cycle de vie des composants défini par les étapes de conception, chargement des composants, exposition des services, invocation et destruction des composants, et un cycle de vie des contrats est donc aussi défini (définition, inscription, application, terminaison, suppression). Par exemple, les contrats basiques sont vérifiés lors de la compilation. Les contrats comportementaux sont vérifiés lors de l'exécution et ne sont généralement pas modifiés. Par contre, les contrats de synchronisation peuvent varier dans le temps, et plusieurs composants peuvent avoir un même contrat de synchronisation. Pour ce qui est des contrats de qualité, il n'y a pas de descriptions très précises mis à part le fait qu'ils spécifient des paramètres de QdS décrivant les qualités requises et offertes, et que celles-ci peuvent potentiellement être négociées. Ces contrats sont appliqués dans le cadre d'applications distribuées [Lorcy *et al.* 1998]. La structure exacte des contrats n'est pas précisée mais une infrastructure pour les supporter est implémentée comme extension d'un intergiciel existant. Cette infrastructure permet d'attacher, détacher des contrats de composants, modifier certains termes des contrats, etc.

CoCoNut

Reussner [Reussner 2001, 2003] propose un modèle pour spécifier les aspects relatifs à l'adhérence, la compatibilité et l'adaptation aux contextes des composants, lors de leur utilisation (assemblage, déploiement, configuration) et de leur exécution (invocation des services requis/fournis). Pour cela, il introduit de nouvelles interfaces fournies et requises qui intègrent en plus des protocoles fournis et requis. Ces protocoles sont exprimés par des machines à états finis et décrivent des séquences d'appels valides, reçus et émis par les composants. Ils peuvent alors être utilisés pour effectuer des tests d'interopérabilité et de substituabilité entre deux composants, par extension des règles de sous-typage de contrat et d'interopérabilité sur les pré et postconditions des méthodes des composants. De plus, des contrats paramétriques sont aussi définis pour lier les interfaces fournies et requises d'un même composant. Ils paramétrisent les préconditions d'un composant par rapport à ses postconditions et vice-versa. Ainsi, ces contrats paramétriques permettent de calculer des cas d'interopérabilité plus faibles entre deux composants en réduisant, par intersection des protocoles, leur coopération aux seules interactions requis-fournis satisfaites. Ces propositions visent et sont expérimentées sur la plate-forme à composants des *Enterprise JavaBeans*.

Contrats de besoins/assurances

Rausch [Rausch 2000] propose un système permettant de décrire les besoins et les assurances des composants (*requirements/assurances* an anglais). Dans ce système, les *besoins* expriment les propriétés dont un composant requiert de son environnement alors que les assurances expriment les propriétés assurées par un composant à son environnement, dès lors que ses besoins sont satisfaits. Ainsi, ces contrats de besoins/assurances permettent de spécifier les *dépendances* externes entre composants, en indiquant pour chacun d'entre eux les assurances qui sont garantis par les besoins satisfaits. Ils sont formés entre des composants et incluent les instances de composants concernées, leur connexions et les correspondances explicites entre les interfaces fournies et requises impliquées dans les relations de besoins/assurances.

Les contrats de Rausch sont spécifiés dans un modèle formel mais les transformations de ce modèle vers les contrats comportementaux restent encore vagues. De plus, les vérifications des contrats sont opérées manuellement lorsque les composants changent, par exemple par des preuves à établir sur les prédicats assurés et requis.

Contrats dans ConFract

Le système *ConFract* [Collet *et al.* 2005] applique l'approche contractuelle aux composants logiciels hiérarchiques. Il définit trois types de contrats permettant de spécifier et de vérifier diverses propriétés sur les connexions des interfaces, les compositions internes et les usages externes des composants. Le contrat d'*interface* contractualise une relation client-serveur, comme les contrats usuels client-serveur dans les systèmes à objets. Il est établi sur chaque connexion entre une interface requise et une interface fournie et regroupe les spécifications des deux interfaces. Le contrat de *composition externe* contractualise l'usage d'un composant vis-à-vis de son environnement, et est construit à partir des spécifications qui ne référencent que des interfaces visibles à l'extérieur de la membrane et placées sur celle-ci. Enfin, le contrat de *composition interne* contractualise, quant à lui, l'assemblage et le comportement d'un composant composite [3] vis-à-vis de son contenu, et ne contient que les spécifications qui référencent des interfaces dans le contenu du composite (interfaces internes du composite, ou interfaces externes des sous-composants). Les contrats sont gérés par une entité dédiée – le *gestionnaire de contrats* – placée sur le composant composite sur lequel est défini le contrat. Comme ils suivent le cycle de vie des composants, ces contrats sont à jour vis-à-vis de l'architecture des composants surveillée, et permettent donc d'effectuer diverses vérifications et localisations d'erreurs. De plus, ils identifient clairement, pour chaque clause de contrat, des *responsabilités* (garant, bénéficiaires, contributeurs), attribuées aux composants par le modèle de contrats. Dans ce système, les contrats sont dynamiquement construits par le gestionnaire de contrats, en fonction des événements d'assemblage des composants, à partir de spécifications exprimées dans un formalisme. De plus, ils sont mis à jour lors des reconfigurations dynamiques. La première version du système intègre un langage d'assertions exécutables nommé *CCL-J* (*Component Constraint Language for Java* en anglais), inspiré d'*OCL* et adapté aux composants logiciels. Ce langage supporte les catégories de spécification classiques (pré et postconditions, invariants de configuration) dont la portée est associée aux interfaces et aux composants. Enfin, les contrats construits sont vérifiés par évaluation, à divers moments, dès que l'environnement nécessaire à leur évaluation est connu. Dans le cas d'un langage d'assertions exécutables comme *CCL-J*, ces moments sont déterminés pour chaque catégorie d'assertions et correspondent, à la phase de fin de configuration des composants pour les invariants ou, à la phase d'exécution, lors des appels et retours de méthodes pour les pré et postconditions. Les clauses des contrats sont donc évaluées selon un environnement d'évaluation bien précis, et permettent de tester qu'un composant respecte bien sa spécification (test d'admission), et de vérifier et détecter les violations de contrats à l'exécution.

La deuxième version du système *ConFract* est en cours de finalisation [Collet *et al.* 2007b; Ozanne 2007] et propose un canevas de contractualisation plus général, nommé *Interact*, qui est abstrait, à la fois, des entités architecturales (composants, services...) et des formalismes de spécification. Il offre ainsi la possibilité d'intégrer des formalismes autres que les assertions exécutables (protocoles comportementaux, logiques temporelles, etc.) [Collet *et al.* 2006]. Un modèle central de contrats est défini dans lequel les spécifications sont réifiées et interprétées, en terme de données échangées entre un composant et son environnement, selon le paradigme hypothèse/garantie (*assume/guarantee* en anglais) [Abadi et Lamport 1993] De plus, chaque contrat construit se compose maintenant de clauses de contrats et d'un accord, tout en continuant à attribuer des responsabilités précises aux entités contractualisées. Ainsi, en utilisant les

3. Un composant composite est un composant formé d'assemblages d'autres composants.

descriptions d'architecture de ces entités, les clauses et l'accord peuvent alors être vérifiés pour évaluer, respectivement, la *conformité* d'une spécification vis-à-vis de son implémentation, et la *compatibilité* entre des spécifications de composants, lorsqu'ils sont impliqués dans une relation horizontale (relation de type client-serveur) ou verticale (relation d'assemblage).

2.3 Contrats électroniques

Plus récemment, avec la maturité d'Internet et le développement des web services, les systèmes monolithiques et les services électroniques classiques (*e-business*) sont progressivement remplacés par un environnement dans lequel des fournisseurs et consommateurs de *services*, faiblement couplés, sont interconnectés et interagissent de façon dynamique. Ainsi, les services sont découverts et établis en fonction des besoins ; les parties qui interagissent sont liées à la demande ; les termes des interactions peuvent changer d'une interaction à l'autre ; les durées des interactions sont variables et celles-ci peuvent s'opérer sous diverses conditions de qualité. Dans ce cadre, de nombreux travaux autour des *contrats électroniques* ont été réalisés. La suite présente la nature de ces contrats ainsi que les systèmes proposés pour les gérer.

Nature des contrats

Les contrats *électroniques* [Merz *et al.* 1998b; Keller *et al.* 2002] ont été proposées pour formaliser les relations entre les différentes parties qui interagissent dans les interactions de service. Ils permettent de formaliser et leur construction et leur gestion de façon automatisée. Les termes spécifiés dans ces contrats sont assez basiques et restent assez similaires à ceux contenus dans les contrats usuels [Merz *et al.* 1998b; Hoffner *et al.* 2001]. Ils reprennent et formalisent les différentes parties du contrat, leurs rôles ainsi que les droits et devoirs de chacune d'entre elles en termes de livrables, de coûts, délais, niveaux de service, paramètres de QdS, et aussi, dans certains cas, des pénalités en cas de non respect des conditions.

Les premières transactions électroniques, basées sur XML et définies dans le cadre de la suite des standards *ebXML* [ebXML 2001], introduisent des contrats qui permettent de formaliser le contexte d'interopérabilité des transactions entre organisations. La construction des contrats s'appuie sur un protocole de collaboration entre les parties nommé *CPPA* (*Collaboration Protocol Profile and Agreement* en anglais) [ebXML CPPA 2002] qui permet de mettre en correspondance et d'affiner successivement les profils des organisations (identités et rôles des organisations, nature des messages échangés, mécanismes de transport des messages...).

Plus récemment, le canevas *WSLA* [Ludwig *et al.* 2003] proposé par IBM permet de définir et de gérer des contrats de QdS sur des services web. Il fournit un langage de spécification de contrats de service qui permet d'associer des contraintes extrafonctionnelles à des descriptions de services exprimées dans des langages dédiées (WSDL [Christensen *et al.* 2006] pour les web services, JSDL [Anjomshoaa *et al.* 2005] pour les soumissions de jobs, etc.). Dans *WSLA*, les contrats de qualité sont exprimés par des *SLA* (*Service Level Agreement*). Un *SLA* représente un type de contrat de Qualité de Service utilisé pour spécifier de façon générale les qualités fournies par un fournisseur à un client. Il spécifie des parties, des descriptions des services (opérations concernées, paramètres de SLA) et des obligations (garanties de QdS, actions et pénalités en cas de violation). Dans *WSLA*, les contraintes extrafonctionnelles portent surtout sur des éléments liées à la performance (temps de réponse, débit, etc.), et au delà de la définition de contrats de qualité, le canevas met surtout l'accent sur la définition d'une architecture pour intégrer des contrats à des interactions basées sur des services en définissant des services de déploiement, de surveillance, de mesures et d'évaluation des contrats [Keller et Ludwig 2003]. D'autres travaux en cours

[Skene *et al.* 2004] tentent aussi de définir de façon plus précise la sémantique des SLA par le biais d'un vocabulaire dédié issu de cas d'étude industriels et une modélisation en *UML* de la syntaxe.

WS-Agreement [Andrieux *et al.* 2005] est un autre standard en cours de définition par dans un groupe de travail du Global Grid Forum. Ce standard fournit des moyens pour définir des accords entre deux parties sur des niveaux de qualité, et il trouve son intérêt dans les situations où les consommateurs de service souhaitent avant d'interagir avoir certaines garanties de conditions de services de la part des fournisseurs (e.g temps de réponse inférieur à 1 seconde pour taux d'arrivées de 100 requêtes à la minute). Structurellement, un accord se forme d'un nom, d'un contexte et de termes qui décrivent à la fois les services (fonctionnalités) et les garanties de qualité sur chaque service. Ces garanties de qualité sont souvent exprimées par des valeurs constantes ou des intervalles de valeurs acceptables, et pour offrir de telles garanties, les fournisseurs sont alors amenés à effectuer de la réservation de ressources selon les demandes des consommateurs. La spécification WS-Agreement spécifie un schéma XML pour décrire les documents XML des accords, une représentation des modèles d'accords (*agreement templates*), des interfaces pour construire et gérer les instances d'accords (surveillance, etc.), ainsi qu'un protocole pour la création d'un accord en utilisant des modèles de contrats (*templates*). Toutefois, bien que cette spécification couvre de nombreux autres aspects, elle ne prend pas encore en compte les possibilités de re-négociation des SLA établis [Andrieux *et al.* 2004; Dan *et al.* 2004].

Les contrats électroniques sont souvent établis de manière unilatérale par le fournisseur de services sans qu'il y ait d'échanges visant à former le contrat d'une façon commune. De plus en plus, le contrat effectif est le résultat d'une phase de *négociation* simple basée sur des échanges successifs de propositions de contrats (*contract template*) [Ludwig 2002; Ludwig *et al.* 2005]. Dans ce cas, les fournisseurs offrent un ensemble prédéterminé de services sous diverses conditions éventuellement ouvertes et les consommateurs choisissent ou complètent les propositions de contrats parmi ceux disponibles. Les processus de négociation mis en œuvre restent spécifiques au système de gestion des contrats associé, et utilisent souvent des techniques de mises en correspondance (*matchmaking* en anglais) des profils des fournisseurs et des consommateurs. Pour élaborer des protocoles de négociation plus sophistiqués, des travaux issus des systèmes à agents sont actuellement repris pour être adaptés au cadre des interactions de services [Andreoli et Castellani 2001; Picard 2003; Kraus 2001].

Bien que les spécifications de SLA ne l'imposent pas, les garanties de qualité des SLA sont, dans la plupart des cas, définis de façon rigide par des constantes [Austin *et al.* 2004; Mobach *et al.* 2005], et la vision générale actuelle est que les niveaux spécifiés doivent être maintenus. Toutefois, il est maintenant reconnu que des possibilités de re-négociation doivent aussi faire partie intégrante des SLA [D'Arienzo *et al.* 2002; Czajkowski *et al.* 2002, 2003] pour par exemple prendre en compte de nouvelles garanties de services (délais de traitements, ressources mémoire et bande-passante, coûts,...).

Systèmes de gestion des contrats

Pour pouvoir gérer les contrats établis, des systèmes de gestion des contrats sont proposés. La suite détaille les deux principaux systèmes.

Le système *COSMOS* [Merz *et al.* 1998a] définit une architecture de gestion des contrats électroniques qui s'appuie sur les entités suivantes : un catalogue qui regroupe et donne accès aux services des fournisseurs et à leurs conditions d'exécution, un courtier (*broker*) qui permet de lier des clients à des fournisseurs, un support de négociation qui permet de construire les contrats par échanges de documents structurés *XML*, et enfin un service de signature qui permet de sceller le contrat établi. Les fournisseurs de services sont implémentés par des CORBA Business Objects et ils publient leurs services en enregistrant leurs adapteurs dans le catalogue. La négociation consiste alors à *i)* modifier les valeurs de QdS, *ii)* modifier les types et références des adaptateurs ou encore *iii)* transférer le contrat entre les parties. Le modèle de contrats défini dans *COSMOS* lui est spécifique et il décrit *i)* les parties (avec leur rôles

et leur identité), *ii)* les obligations des parties en terme de services et de qualité de service exprimée sous la forme d'une liste de couples (attribut de qualité, valeur), *iii)* les relations entre les obligations qui précisent par exemple les moments auxquels réaliser les services ou les traitements à appliquer en cas d'obligations rompues, et *iv)* des aspects légaux qui permettent d'intégrer des clauses juridiques.

Le canevas *Cremona* [Keller *et al.* 2002; Keller et Ludwig 2002] définit une architecture permettant de spécifier, déployer et surveiller les contrats électroniques selon la spécification de *WS-Agreement*. Le système *Cremona* fournit des implémentations des interfaces, définies dans la spécification *WS-Agreement*, afin d'implémenter les fonctionnalités de construction et de gestion des accords établis. En plus, il définit aussi deux types d'entités spécifiques. Le premier est utilisé lors des échanges des modèles d'accords et permet d'accepter ou non les propositions échangées en évaluant les ressources disponibles. Une fois l'accord établi, le second permet de surveiller, les paramètres de QdS et aussi de déclencher des activités pour résoudre les problèmes. Les mesures et vérifications des paramètres de QdS se font par interception des appels entre les clients et fournisseurs, et les actions d'adaptation en cas de violations des contrats peuvent consister à exécuter des actions/politiques prédéfinies, alerter un administrateur ou encore signaler la violation au fournisseur ou au client impliqué. Enfin, il est à noter qu'en dehors de tout aspect contractuel, d'autres travaux [Sayal *et al.* 2003] visent aussi à proposer des plates-formes pour définir et mesurer des métriques de qualité dans les web services (définition de métriques de haut-niveau, séparation de la définition des métriques de leurs mesures, réutilisations des métriques, etc). Des modèles de qualité sont définis pour cela, et ils définissent souvent des méthodes qui permettent de décomposer des métriques basiques en métriques plus complexes.

2.4 Synthèse

Les éléments spécifiés dans les contrats concernent en premier lieu les signatures des opérations ainsi que leurs sémantiques opérationnelles, usuellement décrites par des pré et postconditions ainsi que des invariants. En plus de ces éléments basiques, le protocole et le comportement des composants sont les aspects les plus couramment adressés pour décrire les comportements dynamiques des composants (interactions valides avec un composant vu de l'extérieur, vision dynamique de l'exécution interne d'un composant). Ils sont formalisés par des logiques temporelles [Han 1998], des machines à états finis [Reussner 2001] ou des spécifications de composants en boîtes grises [Buchi et Weck 1997], voire des réseaux de Petri [Giese 2000] ou encore des algèbres de processus [Canal *et al.* 2000]. Vis-à-vis de notre problématique, il en ressort que les modèles et systèmes de contrats proposés concernent surtout les aspects fonctionnels des composants, en dépit des recommandations formulées dans la plupart des travaux pour davantage prendre en compte les spécifications et contrats extrafonctionnels. Beugnard et al. [Beugnard *et al.* 1999] listent des exemples de propriétés extrafonctionnelles telles que la latence, la précision, etc. et évoquent, de façon très générale uniquement, les contrats de qualité et leurs possibilités de négociation. La difficulté de spécifier les propriétés extrafonctionnelles est donc claire, et résulte notamment de celles liée à leur définition et à leur quantification.

Du point de vue de leurs objectifs, les divers modèles de contrats proposés dans les objets et les composants permettent de vérifier les signatures des opérations, la capacité de deux composants à interagir (vision statique et dynamique), la spécialisation de comportement (*refinement* en anglais) [Buchi et Weck. 1999] et la substituabilité d'un composant (*substituability* en anglais) [Reussner 2001]. De ce fait, la plupart des techniques contractuelles se basent sur des approches formelles et se concentrent sur la vérification statique des contrats. Toutefois, comme il n'est pas possible de formellement garantir qu'une implémentation respecte bien sa spécification, les comportement réels lors de l'exécution peuvent malgré tout être différents de ceux attendus. De plus, ces techniques formelles manquent souvent de capacités de *mesure* et de *quantification*, nécessaires pour prendre en compte les éléments liés à l'exécution. Les

vérifications des contrats et la détection d'erreurs à l'exécution peuvent néanmoins se faire par insertion d'assertions dans le code [Meyer 1992a], ou par ajout d'intercepteurs dédiés dans des infrastructures existantes (intercepteurs dans CORBA [Diakov *et al.* 2000; Jin et Han 2005], gestion par des *wrappers* [Mingins et Chan 2002],...). Il faut néanmoins noter que les mécanismes de vérification de contrats à l'exécution ne sont pas l'objet d'un intérêt majeur. Dans cette optique, l'approche contractuelle est souvent délaissée au profit des approches basées sur la collecte et l'analyse de traces d'exécution [Andrews et Zhang 2000].

Le système *ConFract*, quant à lui, se distingue des autres propositions dans le fait qu'il définit différentes formes de contrats, pour spécifier les interactions client-serveur entre composants mais aussi pour prendre en compte d'autres aspects liés à l'assemblage et à l'usage des composants. De plus, comme il réifie les spécifications dans des objets contrats qui maintiennent à jour le contexte nécessaire à leur évaluation, les contrats peuvent être évalués et vérifiés lors de l'exécution à des moments bien précis, et ils permettent ainsi d'effectuer de la détection d'erreurs.

Pour ce qui est des contrats électroniques, leur cycle de vie se décompose globalement selon les phases suivantes. Une première phase dite d'information permet de découvrir, localiser et comparer les services disponibles au travers d'un catalogue de services. Une deuxième phase de négociation consiste à établir les termes des services ainsi que les conditions de leur réalisation. Sous une forme plus ou moins explicite, cette phase aboutit à la construction des contrats, et les ressources demandées sont garanties par réservation. Enfin lors de la phase d'exécution, les services régis par les contrats sont exécutés et surveillés par les systèmes de gestion des contrats. Les contrats sont souvent bilatéraux et reprennent les éléments classiques des contrats usuels. La plupart des contrats électroniques spécifient maintenant les contraintes de qualités par le biais des SLA, et les objectifs de qualité (SLO) sont souvent exprimés par des valeurs constantes ou des intervalles de valeurs acceptables. Les termes des contrats sont soit fixes et imposés par les fournisseurs de services, soit construits par des mécanismes de négociation simples qui consistent en des échanges de propositions de contrats combinées à des mécanismes de mise en correspondance purement syntaxique des SLA. A l'exécution, les contrats sont surveillés par des infrastructures de gestion spécifiques. Celles-ci définissent leur propre modèle de contrats ainsi que leurs mécanismes de gestion dédiés. Concernant la négociation, les mécanismes existants portent surtout sur l'établissement des contrats. Les violations de contrats sont gérées par l'infrastructure sous-jacente et consistent en des générations de rapports de signalisation, enregistrements des pénalités ou encore déclenchements d'actions correctives.

3
Aspects extrafonctionnels

À la différence des aspects fonctionnels qui décrivent très précisément ce que les systèmes réalisent en termes de services et de fonctionnalités rendus, les aspects extrafonctionnels décrivent plutôt les *conditions* dans lesquels les fonctionnalités sont rendues. Ils sont donc importants et doivent être gérés. Toutefois malgré cette importance, comme les aspects extrafonctionnels sont essentiellement décrits par opposition aux aspects fonctionnels, ils ont vocation à englober tout ce qui n'est pas fonctionnel et ne sont donc pas décrits aussi précisément que les aspects fonctionnels. Ces aspects extrafonctionnels sont donc très divers. Ils peuvent être abordés sous *différents points de vue* et il en résulte une très grande *variété* lorsqu'il s'agit de les définir (« à quels aspects s'intéressent-on ? », « quel sens leur donne t-on ? »), de préciser les besoins attendus (« quels sont les besoins ? », « qu'entend-on par bonnes qualités ? ») ou encore de les gérer (« comment établir et maintenir ces qualités ? »).

Dans ce contexte, de nombreux travaux ont été effectués, et on constate que traditionnellement bon nombre d'entre eux ont été réalisés à l'intérieur de communautés qui œuvrent dans des domaines bien précis. Les aspects extrafonctionnels y sont abordés sous certains points de vue très particuliers et les propositions sont donc influencées par les traditions et les expertises de chaque communauté. Une première variation sensible apparaît déjà au niveau de leur *définition*. Par exemple, la notion de *performance*, initialement introduite par la communauté du temps réel, exprime exclusivement des contraintes liées aux temps d'exécution, alors que dans le domaine du génie logiciel, cet aspect désigne plutôt une capacité à réaliser des fonctions sous des contraintes plus larges telles que la rapidité, la qualité des résultats ou encore la consommation en ressources. De façon similaire, la Qualité de Service, notion très ancienne qui a toujours été sous-jacente pour décrire les relations clients-fournisseurs dans le domaine des réseaux, est revisitée et tend à se structurer sous des problématiques et des formes diverses dans le contexte des systèmes électroniques sur Internet, des applications réparties et des systèmes temps-réels.

En plus de cette forte hétérogénéité et de la variabilité des définitions, une autre dimension importante qui s'impose lorsque l'on prend en compte les aspects extrafonctionnels réside dans le fait que ces derniers sont intrinsèquement liés à la notion de *variation*. D'une part, comme les besoins évoluent et que les systèmes ont une durée de vie plus longue, les exigences de qualité définies sur les systèmes ne sont pas les mêmes et doivent être fréquemment reconsidérés. D'autre part, comme les systèmes ont aussi une durée d'exécution allongée, la partie importante des aspects extrafonctionnels qui décrivent le comportement du système peuvent varier fréquemment lors l'exécution et le plus souvent de façon

non-déterministe. Comme ces aspects déterminent les *qualités* du système lors de son fonctionnement, leur gestion devient alors primordiale, et de nombreux travaux ont proposé, à cet effet, divers modèles et mécanismes permettant de les gérer. Dans le domaine des télécommunications et des réseaux, il s'agit de contrôler l'allocation de ressources sur des éléments de l'infrastructure physique ou de définir des protocoles et des politiques de contrôle des données. Dans les intergiciels, l'objectif est de garantir un niveau de QdS de bout-en-bout et de manière transparente en se focalisant surtout sur des propriétés liées aux qualités temporelles des messages échangés. Enfin, dans le domaine du multimédia et du temps-réel, les travaux se concentrent respectivement sur la satisfaction au mieux et la garantie des contraintes temporelles. Les propositions s'intéressent une nouvelle fois aux notions de qualité de services de façon spécifique aux domaines et elles définissent leurs propres mécanismes de gestion sans réellement aborder la gestion des aspects extrafonctionnels dans une vision globale depuis la définition jusqu'à l'utilisation de ressources a niveau de l'infrastructure sous-jacente.

En effet, pour aborder la définition et la gestion des aspects extrafonctionnels de façon complète, ceux-ci doivent idéalement se structurer selon les étapes suivantes :

- la **spécification** des propriétés extrafonctionnelles. Elle consiste à capturer les besoins extrafonctionnels des applications. Pour cela, des langages de modélisation sont utilisés et permettent de définir et de modéliser les concepts extrafonctionnels étudiés avant de les associer à des contraintes extrafonctionnelles. Les spécifications extrafonctionnelles peuvent se situer à différents niveaux, mais elles sont habituellement définies à un niveau d'abstraction élevé et décrivent souvent de façon quantitative des propriétés de haut-niveau liées au comportement externe du système (utilisation, services de haut-niveau, etc.).
- l' **exécution** des aspects extrafonctionnels qui regroupe tout ou partie des étapes suivantes :
  - la **transformation** des aspects extrafonctionnels de haut-niveau en propriétés extrafonctionnelles plus précises. Cette étape sert à exprimer les aspects extrafonctionnels de haut-niveau et à les décomposer en fonction de sous-aspects extrafonctionnels plus précis qui elles peuvent être liées aux éléments de l'application ou de l'infrastructure. Cette étape est optionnelle et dépend du degré d'abstraction des concepts extrafonctionnels et des langages de spécifications manipulés.
  - la **réservation** des ressources. Elle est généralement effectuée avant l'exécution du système dans l'objectif de *garantir* au système des niveaux de ressources suffisants. La réservation des ressources s'effectue après dimensionnement des ressources, et les niveaux de qualités garantis sont souvent établis par un processus de négociation simple qui a lieu de façon automatisée ou non entre les fournisseurs de l'infrastructure et des services, et les clients finaux. Lorsqu'elle est possible, la réservation des ressources est coûteuse et la difficulté essentielle consiste à trouver le bon niveau de ressources à réserver entre d'une part la sous-réservation (inacceptable pour le client) et la sur-réservation (non rentabilisée par le fournisseur). Par ailleurs, cette phase de réservation des ressources ne concerne essentiellement que les aspects extrafonctionnels les plus orthogonaux. Les autres propriétés extrafonctionnelles moins orthogonales et qui sont davantage liées aux fonctionnalités rendues (qualité des services, bonne exploitation des ressources, etc.) ne sont pas directement abordées à cette étape.
  - la **surveillance** et le **contrôle** des niveaux de qualité spécifiés. L'objectif est de contrôler l'exécution du système dans le but de maintenir les niveaux de qualités spécifiés dans les cas où ces derniers n'aient pas été garantis. Pour cela, des mécanismes de *surveillance* des propriétés de qualités spécifiées combinées à des mécanismes et des politiques de *contrôle* sont usuellement mises en œuvre. Ils peuvent par exemple consister à redéfinir des niveaux de qualité alternatifs dégradés ou à adapter certains éléments du système. Dans la littérature, le terme *négociation* correspond plus à l'établissement de niveaux de qualités alors que les mécanismes employés pour maintenir les niveaux préalablement négocié sont généralement inclus sous le

terme d'*adaptation*. Comparé aux mécanismes de réservation de ressources, les mécanismes d'adaptation sont devenus particulièrement pertinents dans le contexte actuel où les environnements sont de plus en plus hétérogènes et de plus en plus changeants.

Ces étapes décrivent un cadre général qui permet d'aborder les aspects extrafonctionnels d'une façon complète. Toutefois, peu de travaux réalisent complètement ces différentes étapes. Au contraire, ils se focalisent sur certaines d'entre-elles et proposent soit i) la spécification et la réservation de ressources à la conception du système, si l'objectif est de définir les besoins pour orienter la conception et l'implémentation avec des garanties de qualités du système résultant à l'exécution, soit ii) de traiter la surveillance et le contrôle et l'adaptation des qualités à l'exécution du système si l'objectif consiste plus à se focaliser sur l'exécution du système pour réagir aux variations de QdS par des adaptations.

La suite de ce chapitre fournit un panorama des différentes dimensions autour des aspects extrafonctionnels. La section suivante revient sur le problème des définitions hétérogènes des aspects extrafonctionnels et a pour but de donner un tour d'horizon rapide des propositions sur ce sujet. Pour mieux appréhender la forte diversité des aspects extrafonctionnels, la seconde section fournit, à partir d'une analyse des travaux existants, différents critères que nous avons établis de façon à permettre de mieux classer les aspects extrafonctionnels. Par la suite, les sections suivantes, en liaison plus directe avec nos travaux, établissent une étude des différentes propositions autour de la spécification et de la gestion des aspects extrafonctionnels. Compte tenu de notre problématique générale (cf. section 1.1), l'accent sera en particulier mis sur les mécanismes de gestion des aspects extrafonctionnels dans les systèmes à composants et lors de l'exécution des systèmes.

3.1 Normes et catalogues généraux

Pour faire face à la forte variété des définitions des aspects extrafonctionnels et à la diversité des interprétations qui en découle, de nombreuses normes et catalogues, tels que IEEE 1061, ISO 9126 ou le catalogue du Mitre, sont proposés avec pour objectif de décrire de façon plus universelle les qualités du logiciel. Ainsi, la norme IEEE 1061 [IEEE 1061 1998] définit et décompose six facteurs de qualité qui concernent l'utilisation du logiciel (utilisabilité), les qualités du logiciel en terme de prise en main (fonctionnalité) et de qualité des services (efficacité, fiabilité) et enfin l'évolution du logiciel (maintenabilité, portabilité). Cette norme vise surtout à fournir une première décomposition des six facteurs de qualités précédents de façon à mieux les décrire. Toutefois, elle ne prend en compte que des aspects définis à un niveau assez général et qui concernent différentes phases du cycle de vie du logiciel. Une deuxième norme ISO/IEC 9126 [ISO/IEC 9126] s'intéresse aux six même concepts de qualités abordés dans la norme IEEE 1061 (utilisabilité, maintenabilité, portabilité, fonctionnalité, efficacité, fiabilité). Toutefois, bien qu'elle se concentre sur les six mêmes facteurs de qualité, ces derniers sont tous décrits sous des décompositions différentes. Pour exemple, le concept de *fiabilité* se décompose dans la norme IEEE selon la capacité à ne pas être défaillant, la tolérance aux erreurs, et la disponibilité, alors que dans la norme ISO/IEC, elle est exprimée selon la maturité, la réparabilité et la tolérances aux fautes. Les autres facteurs de qualité étudiés dans ces deux normes ne sont pas plus détaillés ici. Le lecteur intéressé pourra se reporter aux documents référencés pour des descriptions plus complètes. Un autre catalogue [Clapp et Stanten 1992] proposé par l'organisation du Mitre [Mitre Corporation (Web Site)] propose des définitions d'autres aspects extrafonctionnels qui concernent des propriétés liées à l'utilisation et au fonctionnement (utilisabilité, correction, vérifiabilité, integrité, interopérabilité, résistance aux pannes, efficacité, fiabilité), aux qualités de développement (flexibilité, portabilité) et aux aspects de maintenance et d'évolution (maintenabilité, capacité à faire évoluer, réutilisabilité).

Les aspects extrafonctionnels présentés dans ces normes et catalogues sont des facteurs de qualité de haut-niveau. Ils décrivent différentes propriétés qui concernent le cycle de vie du logiciel (déploiement,

installation, fonctionnement et évolution) et proposent un premier niveau de décomposition. Ces propositions permettent surtout de définir un cadre qui fournit un premier niveau de décomposition des aspects extrafonctionnels mais n'abordent pas les problématiques liées à leur gestion opérationnelle (spécification, vérification, mesures, etc.). Par ailleurs, des différences au niveau de la définition ou de la décomposition des propriétés abordées existent déjà et illustrent l'hétérogénéité mentionnée en introduction de ce chapitre. Les normes et catalogues présentées constituent des alternatives à la définition des facteurs de qualité, et il n'y finalement pas de système de classification unifié des aspects extrafonctionnels et des définitions très précises.

De façon spécifique au génie logiciel à base de composants, un catalogue permettant de décrire les attributs de qualité issus de divers domaines et transposables aux composants logiciels [Brahnmath *et al.* 2002] à été proposé dans le cadre du projet *UniFrame* [UniFrame Project (Web Site)]. Ce catalogue contient une description plus détaillée des qualités utilisables pour les composants logiciels ainsi que des métriques et des méthodologies pour leur évaluation. De plus, les attributs de qualité sont classés selon différents critères : i) le domaine dans lequel ils sont le plus pertinent, ii) leur caractère constant ou dynamique, pour faire la distinction entre les qualités dont les valeurs sont constantes ou variables par exemple lorsqu'elles sont fonction de l'environnement, et iii) leur nature, déterminée selon qu'ils traitent des aspects relatifs à des notions temporelles (e.g délai), d'importance (e.g priorité, précédence), de performance (e.g débit, capacité), d'intégrité (e.g précision), de sécurité, ou encore à des notions plus auxiliaires (portabilité, maintenabilité). Actuellement, différents paramètres de QdS sont présentés dans ce catalogue et synthétisées dans le tableau 3.1. Contrairement aux propositions précédentes, les propriétés extrafonctionnelles introduites dans ce catalogue sont de plus bas niveau. Elles se focalisent plus sur les qualités de services à l'exécution du logiciel (débit, temps de réponse, priorité,...), et des indications un peu plus explicites sont données pour indiquer les mesures qu'elles représentent. Toutefois, les définitions des concepts extrafonctionnels proposés restent encore assez générales et ne s'appliquent pas réellement au domaine des composants logiciels. En particulier, il n'y pas de méthodologie qui détaille comment ces concepts devraient être spécifiées à la phase de conception des composants et comment elles devraient être gérées à l'implémentation et à l'exécution des composants. L'objectif des travaux menés dans ce projet consiste davantage à tendre vers une définition commune de certains paramètres de qualité, par regroupement et adaptation des propriétés définies dans de nombreux domaines voisins. En conséquence, les mécanismes de mesures et les relations entre ces aspects ne sont pas réellement abordés.

3.2 Critères de classifications

L'étude des principaux travaux sur les normes et les catalogues des qualités du logiciel (IEEE 1061, ISO 9126, catalogue de qualité du Mitre) montre que ces derniers restent très généraux et ne proposent pas réellement de système de classification unifié et suffisamment précis des aspects extrafonctionnels. De ce fait et compte tenu de la difficulté du problème, ceux-ci restent alors souvent imprécis et définis de façon *adhoc*. Dans cette section, nous présentons, à titre de synthèse, quelques critères de classification qui permettent de notre point de vue de mieux distinguer les aspects extrafonctionnels. Pour entamer cette discussion, nous considérerons tout d'abord que les aspects extrafonctionnels sont *des éléments ou des propriétés qui influencent les fonctionnalités rendues par ces entités et que l'on peut apprécier d'une manière qualitative ou quantitative*. En particulier, la dépendance entre aspects fonctionnels et extrafonctionnels peut-être traduite par le fait que les aspects extrafonctionnels influencent les aspects fonctionnels en contribuant à leur **définition**, à leur **amélioration** ou à leur **dégradation**. Pour cette classification, nous analyserons les aspects extrafonctionnels en traitant successivement les points de vue à partir desquels on peut les aborder, le cycle de vie des systèmes, le niveau d'abstraction auxquels ils

Aspects	Description
Tolérance aux erreurs (*dependability*)	Mesure de la probabilité d'un composant à être exempt d'erreurs
Sécurité (*security*)	Mesure de la capacité d'un composant à résister aux intrusions
Adaptabilité (*adaptability*)	Mesure de la capacité d'un composant à tolérer des changements en termes de besoins des utilisateurs et des besoins en ressources
Maintenabilité (*maintainability*)	Mesure de la facilité de maintenance d'un système logiciel
Portabilité (*portability*)	Mesure de la facilité pour migrer un système dans un nouvel environnement
Débit (*throughput*)	Mesure de l'efficacité ou la vitesse d'un composant
Capacité (*capacity*)	Mesure du nombre maximum de requêtes concurrentes supportées par un composant
Temps de réponse, latence (*turn-around time*)	Mesure du temps mis par un composant pour rendre le résultat d'une opération
Contraintes de parallélisme (*parallelism constraints*)	Mesure de la capacité d'un composant à supporter des communications synchrones et asynchrones
Disponibilité (*availability*)	Mesure de la durée pendant laquelle le composant est disponible pour offrir un service particulier
Contraintes d'ordonnancement (*ordering constraints*)	Indication sur l'ordre des résultats rendus
Evolution (*evolvability*)	Indication de la facilité d'un composant à évoluer sur une période de temps
Résultat (*result*)	Indication de la qualité des résultats rendus
Réalisabilité (*achievability*)	Indication de la capacité d'un composant à fournir un service de degré supérieur à celui annoncé
Priorité (*priority*)	Indication du fait qu'un composant puisse fournir des services prioritaires
Présentation (*presentation*)	Indication de la qualité de présentation des résultats rendus

TABLE *3.1 – Synthèse des propriétés extrafonctionnelles introduites dans le catalogue UniFrame.*

sont définis, leur décomposition en d'autres aspects extrafonctionnels.

Points de vue Les aspects extrafonctionnels sont généralement exprimés selon des points de vue différents. En particulier, d'un point de vue totalement *externe* au système, ils sont exprimés selon le point de vue des utilisateurs et décrivent des qualités telles que la facilité d'utilisation, la rapidité d'exécution, etc. Ils sont perçus de façon relativement subjective et interprétés de manière qualitative, et les qualités sont souvent caractérisées par les utilisateurs de manière intuitive selon des niveaux qualitatifs qui correspondent à leurs propres interprétations (*«rapidité excellente, performance bonne,...»*). De façon plus générale, les aspects extrafonctionnels observables de l'extérieur du système peuvent être considérés selon qu'ils concernent le fonctionnement général du système (documentation de l'outil, ergonomie de l'application, facilité d'utilisation, internationalisation,...) ou ses comportements en réponse aux actions bien précises des utilisateurs. En particulier, cette dernière catégorie reprend les qualités des services perceptibles de l'extérieur comme, par exemple, le temps de réponse, l'intégrité ou la précision des résultats, etc. Selon un deuxième point de vue, d'autres propriétés décrivent cette fois-ci des informations *internes* au système, et il est devient alors souvent possible d'exprimer les offres et les besoins extrafonctionnels du système de façon plus détaillée et quantitative, en fonction de ses éléments internes. Sous cette vision interne, les aspects extrafonctionnels peuvent être liés soit *(i)* à des éléments applicatifs pour exprimer des aspects incluant des propriétés internes du système (conditions d'utilisation des éléments, performance de codage/décodage, efficacité des algorithmes), des informations sur l'architecture du système, ou encore des mécanismes et des modes de communication entre les différents éléments du système, soit *(ii)* à des ressources physiques (CPU, mémoire, réseau), pour exprimer les besoins et les consommations du système en ressources.

L'existence de ces différents niveaux d'observations fait clairement ressortir le besoin de mécanismes permettant de transformer les aspects extrafonctionnels d'un niveau à un autre. Par exemple, pour une application de type *Video On Demand* (VoD), il s'agirait de pouvoir lier un critère de qualité perçu au niveau d'un utilisateur (ex : *«qualité excellente», «qualité audio»*), à des qualités du niveau applicatif (ex : *«résolution de l'image», «frame rate»*), jusqu'à éventuellement des paramètres du support de transmission (ex : *«débit» ou «jigue»*).

Cycle de vie des systèmes Mis en relation avec le cycle de vie des systèmes logiciels, les aspects extrafonctionnels peuvent aussi être distinguées selon qu'ils concernent *le déploiement et l'évolution* du système logiciel, ou *les qualités à l'exécution* [Malan et Bredemeyer 2001]. Les aspects extrafonctionnels de la première catégorie sont fortement liés à la phase de développement. Ils sont essentiellement déterminés par certains choix ou certaines caractéristiques établis au moment de la conception et du développement. Par exemple, l'internationalisabilité (adaptation aux différences régionales), l'intégration 'plug & play' de composants, l'extensibilité et la réutilisabilité décrivent des aspects liées au déploiement du logiciel et à l'évolution du logiciel. Par ailleurs, ces derniers ont davantage un impact à long terme, car ils déterminent les qualités d'évolution du produit logiciel et influencent les efforts et les coûts associés aux développements en cours et aux évolutions futurs. À l'inverse, les aspects liés au comportement et à l'exécution du système ont un impact sur le logiciel à court terme et selon le contexte d'exécution. Pour exemple, de telles propriétés décrivent les utilisations du logiciel (utilisabilité, efficacité,...), la qualité de service (temps de réponse du système, efficacité d'un algorithme,...), la communication des composants (bande passante, délai, jigue,...), les qualités du système dans les cas de défaillance (tolérance aux fautes, fiabilité, disponibilité, sureté, sécurité,...) ou encore les consommations en ressources.

Abstraction et compositionnalité Comme nous l'avons précédemment mentionné, les aspects extrafonctionnels peuvent aussi être distingués selon les différents niveaux d'abstraction auxquels ils se

situent. Certains aspects, comme par exemple des propriétés collectées au niveau des ressources, sont directement mesurables, alors que d'autres, en raison de leur caractère plus abstrait, ne sont pas clairement définis et ne peuvent donc pas être mesurés. Les aspects sont donc définis à différents niveaux d'abstraction, et un aspect extrafonctionnel peut être de type *primitif* s'il est entièrement défini et de valeur complètement mesurable, ou de type *compositionnel* lorsqu'il existe une *relation* explicite qui permet de l'exprimer ou de le calculer par rapport à d'autres sous-aspects extrafonctionnels.

Par ailleurs, compte tenu des dépendances pouvant exister entre aspects extrafonctionnels,

En particulier, ces aspects compositionnels peuvent être décomposées selon plusieurs façons incluant au moins : 1) une composition au niveau de la **définition** d'un aspect extrafonctionnel qui consiste à décomposer qualitativement un concept extrafonctionnel en fonction de sous-aspects extrafonctionnels plus précis, et 2) une composition au niveau du **calcul** d'une propriété extrafonctionnelle pour exprimer une relation de calcul qui lui permettrait d'être liée par une formule à d'autres propriétés extrafonctionnelles mesurables. Dans le premier cas, il pourrait ainsi s'agir de définir *la qualité* d'une application de type VoD comme une combinaison de *l'interactivité* et de *la qualité de l'image vidéo*, alors que le second cas permettrait d'exprimer le fait que le *temps de réponse* global se calcule en sommant les *délais des traitements* réalisés par les éléments applicatifs et les *les délais de propagation* sur le support physique.

Dans le domaine spécifique des composants, une classification intéressante a été proposée pour décrire dans quelle mesure des propriétés *compositionnelles* d'un assemblage de composants peuvent se décomposer selon des propriétés des sous-composants de cet assemblage ou d'autres facteurs externes. Cinq types de propriétés compositionnelles sont distingués en fonction des différents éléments qui interviennent dans l'expression de la décomposition [Crnkovic *et al.* 2004; Larsson 2004] :

– les propriétés de *composition directe*. Elles sont définies au niveau d'un assemblage de composants et ne s'expriment exclusivement que selon la même propriété définies sur les sous-composants de l'assemblage. La propriété s'exprime par :

$$P(A) = f(P(C_1), P(C_2), ..., P(C_N)) \quad N \in \mathbb{N} \tag{3.1}$$

avec, A l'assemblage, P la même propriété étudiée à la fois sur l'assemblage et les sous-composants et $P(C_i)$ la propriété P sur le composant C_i.
Un exemple d'une telle propriété est l'empreinte mémoire statique d'un assemblage qui, sous une modélisation simple, est égale à la somme des empreintes mémoires de chaque composant qui le constitue. Il est à noter que, dans cet exemple, la fonction *somme* dépend de la technologie de composants utilisée et de ses mécanismes d'assemblages. Dans cette catégorie de propriétés comme dans celles qui vont suivre, la relation de calcul n'est donc pas explicitée mais ce sont davantage les éléments qui interviennent dans les décompositions qui sont étudiés (propriétés des sous-composants, influences externes tierces,...).

– les propriétés de *composition dérivée* sont une généralisation des propriétés précédentes et considèrent, cette fois-ci, le cas où la propriété d'un assemblage représente plus que la somme de la même propriété sur ses sous-composants, et dépend de k autres propriétés définies sur n composants. La propriété s'exprime de façon matricielle par :

$$P(A) = f \begin{pmatrix} P_1(C_1) & P_1(C_2) & ... & P_1(C_N) \\ P_2(C_1) & P_2(C_2) & ... & P_2(C_N) \\ \vdots & \vdots & \vdots & \vdots \\ P_k(C_1) & P_k(C_2) & ... & P_k(C_N) \end{pmatrix} \quad N \in \mathbb{N} \tag{3.2}$$

avec P la propriété de l'assemblage A et $P_k(C_j)$ la propriété k sur le composant j.
Le délai de bout-en-bout dans un système temps-réel est un exemple d'une telle propriété. Avec une technologie à composants qui définit des composants implémentant des tâches de bas-niveau et

dont les interfaces requises/fournies se projettent naturellement sur des entrées/sorties du système ou des périphériques sous-jacents, on peut considérer que ce délai de bout-en-bout s'exprime selon les propriétés de temps d'exécution au pire (*Worst-Case Execution Time* en anglais) et la fréquence d'exécution, toutes deux définies sur deux composants distincts [Hissam *et al.* 2002].

– les propriétés de *composition directe liées à l'architecture*. Elles s'expriment selon la même propriété définies au niveau des sous-composants de l'assemblage (composition directe) mais aussi font intervenir l'architecture de l'assemblage.
La propriété s'exprime par :

$$P(A) = f(P(C_1), P(C_2), ..., P(C_N), SA) \quad N \in \mathbb{N} \tag{3.3}$$

avec les mêmes éléments décrits pour la composition directe (voir Equation 3.1) et en plus SA l'architecture de l'assemblage. L'architecture est souvent employée pour améliorer certaines propriétés du système résultant sans modifier celles des composants individuels. A propriétés des composants constants, différents styles d'architecture peuvent ainsi être mis en œuvre pour offrir des qualités différentes de l'assemblage résultant. Par exemple, la fiabilité ou le degré de concurrence d'un système peuvent être augmentés en introduisant des *pools* de composants et des mécanismes pour inclure ces composants. Cette catégorie de propriétés traduit bien les influences possibles de certains choix architecturaux sur les aspects extrafonctionnels. En revanche, comme celles-ci restent complexes et difficiles à décrire, il n'y a pas plus d'éléments pour décrire et quantifier l'influence de *l'architecture* sur la propriété de l'assemblage résultant.

– les propriétés de *composition liées à l'utilisation*. Comme les propriétés d'un assemblage peuvent, en plus des propriétés internes des sous-composants, dépendre des profils d'utilisation de l'assemblage, la propriété d'un assemblage soumis à un usage donné peut s'exprimer selon des propriétés de ses sous-composants, elles-mêmes calculées en fonction de l'usage du sous-composant concerné. La propriété s'exprime alors par :

$$P(A, U_k) = f(P(C_1, U'_{1,k}), P(C_2, U'_{2,k}), ..., P(C_N, U'_{N,k})) \quad k, N \in \mathbb{N} \tag{3.4}$$

avec P, la propriété de l'assemblage sous le profil d'utilisation U_k, $U'_{i,k}$ le profil d'utilisation du composant C_i obtenu en décomposant l'usage U_k sur l'ensemble des composants C_i.
Le principal problème de ce type de composition réside dans le fait de pouvoir quantifier cette notion d'usage. En particulier, cela implique de pouvoir calculer les usages des composants dans de nombreux et différents contextes qui restent souvent inconnus, et aussi de pouvoir évaluer l'usage d'un composant individuel indépendamment de son contexte d'utilisation dans l'assemblage.

– enfin, les propriétés de *composition liées à l'utilisation et à l'environnement* généralisent la catégorie de propriétés précédente en introduisant en plus l'influence de l'environnement. La propriété s'exprime par :

$$P_k(A, U_k, E_l) = f(P_k(C_1, U'_{1,k}), P_k(C_2, U'_{2,k}), ..., P_k(C_N, U'_{N,k}, E_l)) \quad k, l, N \in \mathbb{N} \tag{3.5}$$

avec sous les mêmes notations, les mêmes éléments définis dans la relation précédente et en plus E_l le contexte d'environnement. Par exemple, les qualités de sécurité d'un système (sécurité, intégrité) dépendent de l'usage du système qui en est fait, mais aussi des risques inhérents à l'environnement. Dans cette catégorie aussi, l'influence de l'environnement est particulièrement difficile à évaluer.

Les catégories de propriétés compositionnelles présentées offrent une classification intéressante et exhibent diverses relations qui peuvent exister entre d'une part les propriétés d'un assemblage et d'autre part les propriétés des sous-composants pris individuellement ou lorsqu'ils sont influencés par les usages, l'architecture ou l'environnement. Les propriétés les plus simples et plus facilement exploitables sont

celles qui appartiennent aux deux première catégories (composition directe et indirecte) car elles ne s'expriment qu'en termes d'autres propriétés internes aux sous-composants de l'assemblage (elles-mêmes facilement mesurables). Les autres catégories ont surtout le mérite d'introduire des facteurs susceptibles d'influencer les propriétés des assemblages (environnement, architecture, usage) mais elles introduisent aussi le problème majeur de l'évaluation de telles influences compte tenu du fait qu'elles ne se calculent ni ne se déduisent automatiquement à partir des (compositions de) composants. A notre connaissance, le problème reste donc difficile et ouvert.

Quantification Pour pouvoir apprécier quantitativement les propriétés extrafonctionnelles de bas-niveau, et faire des observations, celles-ci doivent être mesurées. Lors de ces quantifications, les valeurs mesurées peuvent être constantes ou dynamiques. Les propriétés constantes sont entièrement déterminées et elles restent notamment constantes lors de l'exécution des systèmes (version d'un logiciel, propriétés environnement d'installation). Les propriétés dynamiques fluctuent dans le temps, notamment pendant l'exécution du système (consommation batterie, niveau de CPU, délai, jigue,...). Cette différenciation entre propriétés constante et dynamique, et le degré de fluctuation d'une manière plus générale, dépend toutefois des conditions d'observation et doit notamment être modulée par l'échelle de temps considérée. Par exemple, certaines propriétés définies à la phase de développement comme étant des constantes, ne varient plus à l'exécution du système (version d'un composant, environnement d'installation,...), toutefois, elles le seraient si la période d'observation inclut plusieurs cycles de développement pendant lesquels des évolutions majeures de conception sont réalisées. Cette fenêtre des observations est à définir et à ajuster plus précisément en fonction des besoins et du contexte des éléments observés. De plus, un compromis doit aussi être fait entre des mesures très fines, mais complexes et coûteuses à obtenir (valeurs fortement changeantes et mesurées périodiquement), et des mesures moins précises mais qui peuvent être plus facilement mesurées et sont plus facilement exploitables (mesures statistiques). Ainsi, souvent en plus des valeurs mesurées, des notions d'erreurs sont aussi souvent introduites pour fournir en parallèle une appréciation de la validité des mesures.

Interdépendances Pour avoir une modélisation assez précise des aspects extrafonctionnels, il est nécessaire de considérer les relations qui peuvent exister entre ceux-ci. Comme nous l'avons évoqué précédemment, des relations de composition et de calcul permettent d'exprimer la composition d'un aspect extrafonctionnel selon d'autres sous-aspects extrafonctionnels, mais de façon plus indirecte, il peut aussi exister des *interdépendances* ou influences entre aspects extrafonctionnels moins directement liés. Par exemple, le développement de code natif contribue à accroître les performances du système, mais cela affecte manifestement la portabilité et la maintenabilité du système. Dans la même idée, l'intégration de techniques de cryptographie contribue à garantir un certain niveau de sécurité mais souvent au détriment de performances en terme de consommation de ressources (mémoire, CPU,...). L'utilisation d'adapteurs spécifiques pour augmenter l'interopérabilité entraîne usuellement un surcoût en terme de consommation de ressources. Enfin, l'ajout de réplicas permet d'accroître la disponibilité du système (temps de réponse,répartition de la charge,...) et la fiabilité (tolérance aux pannes) mais contribue aussi en termes de complexité de déploiement multi-sites et de consommation de ressources (mémoire, CPU,...). Il existe donc de façon très claire des interdépendances entre les aspects extrafonctionnels, et celles-ci doivent être prises en compte. Un verrou majeur réside notamment dans le fait de pouvoir exprimer de telles dépendances, avant de pouvoir les quantifier pour celles qui sont mesurables. De façon très grossière, il est déjà possible de classifier de telles dépendances entre propriétés extrafonctionnelles selon qu'elles soient *contributives* (la réalisation des objectifs de qualité des aspects des uns contribue à ceux des autres), *conflictuelles* (les évolutions souhaitées des aspects extrafonctionnels sont inversement proportionnelles). Ce dernier cas est celui qui est le plus souvent rencontré mais aussi le cas problématique, car il s'agit

alors de trouver un compromis optimal dans la réalisation des aspects extrafonctionnels dépendants.

3.3 Langages de spécification

Pendant de nombreuses années, les besoins extrafonctionnels des applications ont été écrites textuellement en tant que partie intégrante des documents de spécification des besoins. Depuis, de nombreux langages ont été proposés pour permettre de modéliser et de décrire les besoins extrafonctionnels des applications tout en les séparant du code métier. Ils s'adressent initialement à une utilisation humaine et présentent notamment des atouts de compréhensibilité et de réutilisabilité supportés par des mécanismes d'héritage, de regroupement ou encore de nommage des spécifications.

Dans le domaine des intergiciels et des infrastructures *CORBA*, les langages *QIDL* [Becker et Geihs 2000] et *QDL* (*QoS Description Language* en anglais) [Pal *et al.* 2000b] ont été proposés. *QIDL* défini dans le cadre bien précis du système *MAQS* (*Management for Adaptive QoS-enabled Services* en anglais) [Becker et Geihs 1997] et permet de spécifier des contraintes de QdS très simples en étendant les descriptions des interfaces, et d'affecter ces contraintes aux interfaces fonctionnelles. *QDL* est un autre langage défini dans le système *QuO*[4] [Quality Objects (QuO) Project (Web Site)] qui permet d'exprimer différents aspects de QdS (contrats de QdS, description des objets et des délégués, configuration des applications). Une caractéristique particulière de *QDL* est qu'il permet, en plus de la déclaration de spécifications de qualité, de définir des règles d'adaptation qui sont définies en combinant les caractéristiques de qualité à certains éléments spécifiques de l'implémentation. Toutefois, malgré cette qualité, l'entrelacement des concepts de qualité avec certains détails de l'implémentation de la plate-forme rend le langage complexe et difficile à appréhender.

Un autre langage *QoSCL* (*QoS Constraint Language* en anglais) [Defour *et al.* 2004b] a été défini récemment pour définir des niveaux de qualité aux interfaces fournies et requises des composants. Le langage s'appuie sur les concepts de qualité, niveau de qualité et dépendances extrafonctionnelles. Les qualités sont contraintes pour définir les niveaux de qualité requis et fournis, et des relations explicites permettent d'exprimer des dépendances entre QdS requises et fournies d'un composant. A partir de ces relations de dépendances, il est alors possible de valider, en phase de déploiement et par un moteur de résolution, les niveaux de qualité des composants individuels, mais aussi ceux de plusieurs composants connectés, par propagation des relations de dépendances.

Dans le domaine des applications multimédias distribuées, le langage de spécifications *HQML* [Gu *et al.* 2001b] basé sur *XML* a été proposé dans l'environnement *QoSTalk* [Gu *et al.* 2001a]. Ce langage reste très spécifique au domaine multimédia. Il reprend les éléments de base des autres langages mis à part le fait qu'il permette de spécifier des aspects extrafonctionnels plus étendus concernant les qualités des utilisateurs, de l'application et des ressources système. De plus, il autorise la définition de règles simples d'adaptation qui permettent de choisir de nouveaux profils de QdS, construits à partir de configurations d'applications possibles.

Le langage *QML* (*QdS Modeling Language* en anglais) [Frølund et Koistinen 1998a,c,b], développé au sein des laboratoires Hewlett-Packard, propose un langage de spécification de qualité plus général. Il est implémenté comme une extension de *UML* et organise la définition des spécifications de QdS selon trois abstractions principales : le type de contrat, le contrat et le profil. Le type de contrat permet de modéliser une catégorie de QdS et il est, pour cela, composé d'un ensemble de dimensions de QdS représentant chacune une propriété extrafonctionnelle définie par, son domaine de valeurs, son unité et le sens de variation souhaité. Les contraintes sont définies sur ces dimensions, et regroupées dans une instance de contrat. Les contrats de qualité correspondent à de simples déclarations de contraintes de qualité qui définissent un intervalle de valeurs valides. *QML* permet aussi de caractériser l'évolution

4. Les travaux de ce projet seront décrits plus loin dans ce document.

des dimensions par le biais de concepts mathématiques statistiques simples comme la moyenne ou la variance. Enfin, le profil permet d'associer les contrats aux interfaces ou aux opérations, et le type de contrat (requis ou offert) dépend directement de type de l'entité (client ou serveur) auquel le profile est associé.

Le listing 3.1 qui suit décrit un exemple de contract type définissant la catégorie extrafonctionnelle de nom `Reliability`, et composée des dimensions `numberOfFailures`, `TTR` et `availability`.

```
type Reliability = contract {
  numberOfFailures: decreasing numeric no/year;
  TTR: decreasing numeric sec;
  availability: increasing numeric;
}

systemReliability = Reliability contract{
  numberOfFailures < 10 no / year;
  TTR{
    percentile 100 < 2000;
    mean < 500;
    variance < 0.3
   };
  availability > 0.8;
}

rateServerProfile for RateServiceI = profile{
  require systemReliability
  from  operationName require anotherContractInstance;
}
```

Listing 3.1 – Exemple de type de contrat, de contrat et de profil en QML.

L'instance de contrat de nom `systemReliability` définit des contraintes sur les dimensions précédentes. Celles-ci sont liées aux interfaces et opérations par le biais du profil `rateServerProfile` défini sur l'interface `RateServiceI` et sur l'opération `operationName`. En plus de ces mécanismes de spécification, *QML* définit aussi un système de représentation des spécifications à l'exécution *QRR* (*QdS Representation Runtime* en anglais) [Frølund et Koistinen 1998c]. Ce système se charge de construire des représentations des spécifications pour qu'elles puissent être gérées à l'exécution. Des structures de données sont ainsi construites pour représenter les propriétés extrafonctionnelles et leurs contraintes. Par cela, le système rend alors possible la vérification des spécifications à l'exécution en évaluant les structures de données par le biais d'interfaces prédéfinies à implémenter. Contrairement à *QuO*, rien dans ce langage ne permet de définir des stratégies d'adaptations. De plus, les spécifications définies dans ce langage restent très peu liées au éléments sous-jacents de l'application.

CQML (*Component Quality Modeling Language* en anglais) est un autre langage de spécification pour QdS proposé par J.Ø. Aagedal [Aagedal 2001]. Ce langage permet de spécifier la QdS offerte et requise dans les systèmes à composants de façon plus générale et plus précise que *QML*. Pour cela, trois éléments de base sont définis (QoS characteristic, QoS statement, QoS profile) respectivement inspirés des abstractions type de contract, contrat et profil de *QML*. Dans *CQML*, la construction de base est la QoS characteristic, défini par un nom unique, un domaine de valeurs (numérique ou ensembliste), et un sens de variation souhaité (croissant ou décroissant). Comparé à *QML*, une différence majeure est que les spécifications extrafonctionnelles en *CQML* peuvent être paramétrées par des attributs de l'application dont les valeurs ne seront connues qu'à l'exécution. Les QoS statements représentent les contraintes sur les concepts de qualité – QoS characteristic – précédents, et les QoS profiles définissent la mise en correspondance des contraintes de niveaux de qualité aux composants. Un composant peut ainsi

soit fournir, soit requérir un QoS statement donné. L'avantage de *CQML* est qu'il est plus adapté aux composants logiciels car il permet d'exprimer les offres et les besoins en qualité des composants, et cela, de façon plus riche, par paramétrisation des spécifications. Toutefois, comme *QML*, les spécifications extrafonctionnelles restent encore peu liées aux composants à l'exécution, et on ne sait pas très clairement comment celles-ci se projettent et se réalisent dans une plate-forme d'exécution. Enfin, *CQML* permet aussi d'inclure différents profils de QdS pour un composant et de définir des règles de transitions entre ceux-ci à opérer sur certains *callbacks*. Cela fourni ainsi un mécanisme d'adaptation par changement de profils de QdS.

Enfin, le langage de spécification *CQML+* [Röttger et Zschaler 2003] s'inscrit directement dans la ligné de *CQML* et a été établi dans le cadre du projet *COMQUAD* afin de mieux décrire le traitement des spécifications de QdS à l'étape d'exécution. Pour cela, un méta-modèle définit notamment des *événements* (e.g appel d'une méthode), des *queues d'événements* (e.g liste des appels de méthodes d'un composant) ainsi que des *ressources* permettant de modéliser des ressources physiques comme le CPU ou la mémoire. Bien que ces travaux permettent d'enrichir *CQML* et d'exprimer plus finement des spécifications extrafonctionnelles, la sémantique n'est toujours pas précise et est à déterminer par le système sous-jacent. Par ailleurs, au lieu de spécifier des domaines de valeurs déterminées statiquement, une clause *QdS_dependencies* est aussi définie pour expliciter par des équations la relation entre une QdS fournie et une QdS requise au niveau d'un même composant [Zschaler et Meyerhöfer 2003]. Ces équations sont des formules exactes ou des formules intégrant des intervalles dont les bornes sont soit fixes ou soit définies par des fonctions. Toutefois, cette dépendance exprime un lien entre la QdS requise de celle fournie sur un même composant mais on ne dispose pas des contributions de plusieurs composants sur une même propriété de qualité.

3.4 Gestion des contrats et adaptation dynamique

Dans cette section, nous présentons les principaux travaux sur l'adaptation dynamique en nous attardant principalement sur *i)* les infrastructures de gestion qui intègrent des contrats, et *ii)* les systèmes d'adaptation définis dans les intergiciels, les applications multimédias et les composants.

3.4.1 Techniques pour l'adaptation

Architecture dynamiques

Les architectures logicielles ont pour objectif de modéliser de façon explicite les entités et les connexions des systèmes distribués. Dans cette idée, les ADL (*Architecture Description Language* en anglais) représentent un moyen couramment employé pour spécifier de telles architectures sous la forme d'un graphe acyclique formé de composants et de connecteurs. Sur de telles architectures, des *styles architecturaux* ou *contraintes* [Medvidovic et Taylor 2000; Moreira *et al.* 2001] sont utilisées pour spécifier des propriétés des architectures ou pour spécifier des logiques d'adaptation décrivant les actions à appliquer pour transformer une architecture donnée en une autre ainsi que les événements les déclenchant [Moazami-Goudarzi 1999; Georgiadis 2002]. La plupart des actions d'adaptation s'opèrent par réécriture d'architecture et mettent souvent en œuvre des primitives de reconfigurations au niveau des composants et des connecteurs telles que `replace-component`, `rebind-connector`, etc. [Moazami-Goudarzi 1999; Georgiadis 2002; Dashofy *et al.* 2002]. De nombreuses approches pour l'auto-adaptation sont basées sur les architectures dynamiques, car celles-ci formalisent l'auto-adaptation et proposent des modèles pour atteindre des reconfigurations cohérentes. Toutefois, elles ne résident souvent qu'au niveau de l'architecture et reposent exclusivement sur une entité dédiée et centralisée qui opère les adaptations au niveau de l'architecture globale [Dashofy *et al.* 2002; Garlan et Schmerl 2002; White *et al.* 2004].

Systèmes réflexifs

Les systèmes réflexifs possèdent une représentation d'eux-mêmes et offrent la possibilité d'obtenir diverses informations qui les décrivent (introspection) et de les modifier (intercession). Pour cela, certains de leurs éléments, habituellement implicites, sont *réifiés* et des méta-modèles, causalement connectés au système de base, sont définis pour manipuler ces réifications. Les méta-objets sont généralement rendus accessibles et modifiables par un protocole à méta-objets et les modifications effectuées sur ces derniers sont répercutées sur le niveau de base. Ainsi, parce qu'ils offrent une représentation et ouvrent les systèmes, les techniques réflexives fournissent un mécanisme très puissant pour facilement mettre en œuvre les adaptations et reconfigurations des systèmes [Blair *et al.* 2002; Redmond 2003]. Toutefois, les techniques réflexives seules ne suffisent pas, et d'autres infrastructures pour gérer la cohérence et l'intégrité des reconfigurations sont aussi nécessaires.

3.4.2 Principaux travaux

Contrats et infrastructures de gestion

JAMUS La plate-forme *JAMUS* (*Java Accommodation of Mobile Untrusted Software* en anglais) [Sommer 2003] définit des contrats pour spécifier et réserver l'utilisation des ressources par les composants. Cette plate-forme permet d'héberger les composants *Java* capables d'exprimer leurs besoins en ressources, et s'appuie notamment sur l'environnement *RAJE* qui fournit des fonctionnalités d'observation et de contrôle des accès aux ressources. Le processus de contractualisation se décompose en une phase de soumission des contrats et de test de réservabilité des ressources, suivie d'une phase d'enregistrement des contrats qui permet de mettre à disposition les ressources réservées. Une entité dédiée centralise la gestion des ressources (demandes, évaluation, surveillance), et forme le contrat lorsqu'un niveau de QdS requis par une application est satisfaite par l'infrastructure hôte, un accord contractuel est alors formé entre les deux parties. De plus, chaque composant s'exécute dans un *conteneur* spécial qui lui permet d'être isolé du reste des composants (espace propre de nommage, absence de partage, absence d'accès à des objets référencés par d'autres composants,...). Ce conteneur prend en charge les ressources réservées par le composant et intègre un moniteur d'application qui extrait les contrats et surveille la correcte utilisation des ressources. Les violations des contrats sont traités par des notifications, éventuellement suivis de sanctions (verrouillage de l'accès aux ressources, etc.).

CR-RIO Un autre canevas, nommé *CR-RIO* (*Contractual Reflective-Reconfigurable Interconnectable Objects* en anglais) [Cerqueira *et al.* 2003; Loques et Sztajnberg 2004] permet de décrire, déployer et gérer les aspects fonctionnels et extrafonctionnels dans les systèmes à composants. Ce canevas définit comme éléments de base, un modèle de composants basé sur des modules, des connecteurs qui définissent les relations entre composants et qui concentrent sur les aspects extrafonctionnels, et des ports qui identifient les points d'accès aux modules et aux connecteurs. Un ADL est aussi défini pour décrire l'architecture des composants, et celle-ci est conservée dans un référentiel et utilisée pour déployer l'application et réaliser des vérifications formelles [Braga et Sztajnberg 2004]. Enfin, un configurateur fournit divers services pour gérer les applications. Les aspects extrafonctionnels sont décrits dans des contrats dits d'architecture qui expriment *i)* les besoins des deux parties liées dans une relation client-serveur, et *ii)* des politiques de négociation. Ces contrats sont exprimés dans un langage de QdS, sous ensemble de *QML* (cf. page 30), et sont gérés à l'exécution par l'infrastructure. En plus des éléments classiques de *QML*, une clause de négociation décrit les adaptations à réaliser lorsque des conditions du contrat évoluent. En l'état, la négociation consiste à établir une autre configuration des composants qui autorise un niveau de qualité dégradé mais les mécanismes mis en œuvre pour déterminer et appliquer cette nouvelle configuration ne sont pas plus décrits. L'infrastructure de gestion des contrats s'appuie sur

une entité globale, qui agit en tant qu'autorité de gestion ayant une vue sur l'ensemble des contrats et des services exécutés, des entités locales, représentant chaque participant client ou serveur, et enfin des *agents de QdS* en charge de la gestion des ressources. Ainsi, l'entité globale identifie le service à négocier et envoie les contrats de QdS aux entités locales de chaque participant, lesquels traduisent les contraintes de QdS en paramètres systèmes gérés par les différents agents de QdS. Ces derniers effectuent dès lors la réservation de ressources et la surveillance des paramètres.

Projet COMQUAD Le projet *COMQUAD* [COMQUAD project (Web site) 2001] (*COMponents with QUantitative ADaptivity* en anglais) définit un modèle de composants et un environnement pour la spécification des aspects extrafonctionnels dans les applications à base de composants. La particularité de ce travail réside dans le fait qu'il propose une approche très complète qui s'appuie sur des notions de langages de spécifications, de contrats et de négociation. De nombreux éléments sont ainsi définis : un langage de spécification des aspects extrafonctionnels, nommé *CQML+*[5], une architecture de composants, des rôles et de types de contrats de QdS, et enfin des mécanismes pour gérer la négociation des contrats et la réservation des ressources. Le modèle de composants est bâti au dessus du modèle des *EJB*. Il sépare clairement les ressources système des composants, et définit différentes entités dédiées qui gèrent, d'un coté les accès aux ressources, et d'un autre, l'exécution des composants. Pour gérer la QdS, différents rôles sont définis (*opérateur*, *système*, *conteneur* et *plate-forme*, et quatre types de contrats permettent de décrire les besoins en QdS des clients, des composants ou du conteneur, et les capacités des composants ou de la plate-forme à en offrir. Ainsi, de manière transitive, les QdS spécifiées au niveau des clients sont traduites en ressources à réserver par la plate-forme.

Le système propose aussi des mécanismes de négociation [Göbel *et al.* 2004; Mulugeta et Göbel 2005] et de réservation des ressources [Baumgartl *et al.* 1998; Härtig *et al.* 1999]. La négociation est initiée lorsqu'un client désire instancier des composants à un niveau de qualité donné. Les niveaux de qualité sont décrits par des structures générées à partir des spécifications *CQML+*, et le processus de négociation consiste à sélectionner une implémentation de composant adéquate, qui fournit les qualités spécifiées par le client à partir d'une liste d'implémentations disponibles, ou alors dans le cas, où aucune implémentation n'est satisfaisante, à choisir une autre profil de QdS spécifiant des contraintes de qualité moins fortes. L'algorithme de négociation teste ainsi toutes les configurations d'implémentations possibles et vérifie, pour chaque connexion de composants, la validité du contrat établi. Un contrat de QdS entre deux composants ne peut être établi qu'à la condition où la QdS requise par un composant lui est fournie par un autre, et si cette condition n'est pas satisfaite alors l'assemblage de ces deux composants est impossible, et le contrat n'est pas construit. Le processus de recherche induit une complexité combinatoire et ne garantit pas l'obtention de la meilleure configuration possible. Des optimisations sont néanmoins proposées pour réduire l'espace de recherche, par exemple, en retirant les implémentations dont la QdS requise n'est fournie par aucun autre composant, ou encore en classant les implémentations par ordre de qualité descendante.

Enfin, la gestion des ressources s'effectue dans l'environnement *DROPS* (*Dresden Real-time Operating System* en anglais) qui effectue la réservation des ressources par des dialogues entre un gestionnaire principal qui reçoit les requêtes de réservation de ressources, et les différents gestionnaires associés aux ressources. Ces dialogues se font par des interfaces et des opérations clairement définies, et les ressources à réserver sont décrites sous la forme de couples (ressource,valeur). Par la suite, chaque gestionnaire de ressource est responsable de la surveillance de sa ressource, et il notifie le gestionnaire principal des problèmes de réservation (ressource invalidée, modification de la ressource...), à charge pour ce dernier de gérer les erreurs.

5. Ce langage a été précédemment décrit dans ce document en section 3.3, 32.

QuO Le projet *QuO* [Quality Objects (QuO) Project (Web Site)] est un canevas qui permet de construire des applications distribuées auto-adaptatives (adaptations de paramètres d'une vidéo dans des applications multimédias distribuées, réglages des paramètres du protocole TCP...) [Atighetchi *et al.* 2003]. Le prototype de base de *QuO* s'appuie sur *CORBA*, et comparé aux mécanismes standards d'invocation qui utilisent un *ORB*, il supporte des contrats de QdS et définit des entités supplémentaires (délégués, objets de surveillance...) qui lui permettent de surveiller et de réagir aux changements du système [Pal *et al.* 2000a]. Le langage *QDL* associé à *QuO* permet de définir des contrats de QdS et se décompose en deux sous-langages qui permettent, respectivement, *i)* de spécifier les niveaux de qualité requis par le client et fourni par le serveur, et *ii)* de spécifier les actions d'adaptations à exécuter en réponse aux changements des états de QdS. Ainsi, dans *QuO*, les contrats de QdS définissent les états de QdS acceptables (régions de valeurs) et les logiques d'adaptations sont spécifiées en indiquant les méthodes à exécuter en réponse à des transitions d'états donnés [Sharma *et al.* 2004]. Plus précisément, les contrats contiennent les différents éléments suivants : *i)* les niveaux de qualités spécifiés, *ii)* les références à des objets de surveillance qui mesurent et contrôlent les niveaux spécifiés, *iii)* l'ensemble des transitions d'états pour déclencher les actions d'adaptations, et *iv)* des objets de *callback* qui permettent d'effectuer des notifications de changements d'états aux objets client ou serveur. Au niveau de son infrastructure de gestion, les niveaux de qualité sont encapsulés à l'exécution dans des objets de contrats, et les logiques d'adaptation sont tissés lors de la génération des objets *délégués* afin que ces derniers puissent effectuer des intercessions sur les appels de méthodes (point d'appel, avant l'appel distant, à l'entrée de la méthode distante, et les retours correspondants). *QuO* fournit aussi des objets dédiés (*syscond* pour *system condition* en anglais) pour mesurer diverses informations de bas niveau sur les ressources du système. Ces objets exposent des interfaces permettant de collecter les informations de différentes manières (simple requête, requêtes périodiques, etc.) et peuvent être invoquées en tant que méthodes d'adaptations. Enfin, il est à noter que ce système ne fournit pas de support pour explicitement gérer la cohérence des actions d'adaptations. Celle-ci reste à la charge du développeur. De plus, pour mettre à jour les logiques d'adaptations, il faut réécrire de nouvelles spécifications en *QDL*, les récompiler et les re-déployer. Enfin, un inconvénient majeur de *QuO* est qu'il est restreint aux relations client-serveur et que les logiques d'adaptations restent relativement complexes à spécifier et à maintenir car le nombre d'états et d'actions d'adaptation à spécifier augmente radicalement.

Systèmes d'adaptation

Adaptation dans les intergiciels Dans le domaine des intergiciels et des systèmes distribués, le but est de fournir un niveau de QdS de bout en bout et de manière transparente en mettant en oeuvre la coopération entre les différentes composantes du système, du support de communication et de l'application. La gestion de la QdS consiste à adapter les offres du système réparti aux besoins des utilisateurs, en prenant en considération différents facteurs (nature du service, support de communication, applications des utilisateurs,...). La plupart des travaux réalisés dans les intergiciels se focalisent sur une infrastructure *CORBA*. Ils s'intéressent surtout au contrôle de la *priorité* et proposent de configurer et contrôler les ressources et les communications par des interfaces standards et des politiques de QdS [Real-time Corba 1997; Schmidt et Kuhns 2000] ou des patrons de conception [Schmidt 2001]. Les techniques mises en œuvre portent sur différents éléments au niveau du système d'exploitation, des couches réseaux ou encore de la communication inter-processus, et s'appuient fortement sur des capacités d'introspection et de reconfiguration du système [Kon *et al.* 2000]. Une synthèse de la gestion des ressources dans les intergiciels est présentée dans [Duran-Limon *et al.* 2004].

Dans un autre cadre, le projet *Erdos* (*End-to-end Resource Management of Distributed Systems* en anglais) propose un canevas qui offre des possibilités de contrôle d'admission et dégradation de QdS dans les systèmes distribués [Chatterjee *et al.* 1999]. Dans ce système, des entités dédiées sont définies pour

gérer chaque ressource à surveiller, l'application donnée et le système entier. Le contrôle d'admission est effectué par le gestionnaire du système qui, à partir d'algorithmes de gestion de ressources, détermine les ressources nécessaires à allouer et coordonne l'utilisation qui en est faite. Ces informations sont alors transmises au gestionnaire de l'application et des ressources. La dégradation des paramètres de QdS n'est évoqué qu'au travers d'exemples, notamment, une application multimédia dans laquelle une adaptation structurelle (redirection vers un proxy qui prend en charge le coût du décodage du flux) et des algorithmes de dégradations pour sur les données multimédia (diminution de la résolution ou du débit des images) sont mis en œuvre.

Open ORB [Coulson *et al.* 2002] définit une plate-forme, sous plusieurs implémentations (COM, CORBA...), qui propose des possibilités de (re)-configuration des applications par des techniques de réflexion. Les différents aspects structurels (accès aux composants et leurs implémentations) et comportementaux (interceptions, ressources) sont organisés sur différents niveaux méta. Les politiques et actions d'adaptations sont représentées par des automates qui sont directement traduits vers les composants à l'exécution, et utilisent les capacités d'introspection de la plate-forme pour agir sur la structure et le comportement des composants [Blair *et al.* 2002].

Adaptation dans les applications multimédia Dans les applications multimédias distribuées, la QdS est abordée de façon plus globale et prend en compte des aspects extrafonctionnels liés au réseau, aux ressources système, aux applications et aux utilisateurs. Les spécifications de QdS concernent ainsi des propriétés très diverses telles que des critères de qualités perceptibles au niveau des utilisateurs (taille de l'image, qualité de l'image...), des caractéristiques temporelles ou spatiales des flux de données (frame rate, taux de compression, résolution...), des éléments du support de communication (délai, ordonnancement des messages...) ou des aspects de synchronisation [Bouch et Sasse 2001; Jin et Nahrstedt 2002, 2004]. La gestion, à proprement dite, se décompose selon les phases usuelles de *i)* traduction des besoins, *ii)* négociation d'un accord initial de qualité, et *iii)* exécution avec maintien ou modification du niveau de QdS [Aurrecoechea *et al.* 1998]. Les variations à l'exécution sont gérées par des systèmes fermés qui intègrent leurs propres règles de reconfiguration. Celles-ci sont souvent statiquement définies et consistent à modifier tout ou partie des éléments de la chaîne de traitement du flux multimédia [Fu *et al.* 2000; Waddington et Coulson 1997].

D'autres travaux proposent de réaliser l'adaptation dynamique en s'appuyant sur des architectures à composants [Black *et al.* 2002; Lohse *et al.* 2002], et s'intéressent de plus en plus aux capacités de reconfigurations dynamiques. Ainsi, le canevas *PLASMA* (*Platform for Self-Adaptive Multimedia Application* en anglais), basé sur des composants hiérarchiques, à été proposé dans le projet *Sardes*, pour construire des applications multimédias auto-adaptatives [Layaïda *et al.* 2004; Layaida et Hagimont 2005]. Ce canevas exploite les capacités hiérarchiques et réflexives des composants *Fractal* et définit trois types de composants : les *composants métiers* qui encapsulent les opérations de traitement des données multimédia, les *composants de sondes et surveillance* qui, respectivement, donnent accès à diverses informations sur les paramètres sur la QdS et les ressources, et surveillent les valeurs mesurées à des seuils spécifiés, et enfin les *composants de reconfiguration*, qui effectuent des actions de reconfigurations. Celles-ci peuvent consister en des modifications de valeurs d'attributs ou en des changements de la structure interne d'un composite mais dans les deux cas, les reconfigurations sont déclenchées par une modification d'attribut. Par ailleurs, comme tous les éléments – applicatifs et de reconfiguration – sont représentés de façon homogène par des composants, et que les actions de reconfiguration se traduisent exclusivement par des modifications d'attributs, un langage spécifique très proche de l'ADL est utilisé pour supporter la description des différents composants métiers, celle des différents composants de surveillance et de reconfiguration ainsi que les observations et les actions de reconfiguration de ces derniers, exprimées sous la forme de changement d'attributs.

Enfin, d'autres travaux [Dalmau *et al.* 2004], menées dans le projet *ALM*, abordent l'adaptation dynamique par un processus qui évalue le contexte et calcule la meilleure configuration de composants. Les caractéristiques des composants individuels sont identifiées, puis, les différentes configurations possibles des composants sont modélisées sous la forme de graphes. La comparaison des qualités de service des différentes configurations se fait alors par injection de valeurs dans les graphes et calcul de la configuration optimale, par un fonction de distance simple. En revanche, il y a peu de détails donnés sur les types de propriétés spécifiées, ainsi que sur les mécanismes employés pour la surveillance et les mesures des caractéristiques de qualité des composants individuellement ou assemblés.

Adaptation dans les composants Dans les systèmes composants, le système *SAFRAN* [David. 2005] (*Self-Adaptive Fractal Components* en anglais) propose un canevas pour l'auto-adaptation dans les composants. L'adaptation est considérée comme étant une préoccupation transverse intégrée à l'application par tissage. Le canevas proposé vise les applications construites à base de composants *Fractal*, et il se base sur deux systèmes. Un système *sensible au contexte* et un système pour le *développement de politiques d'adaptation*. Le service *sensible au contexte* d'exécution fournit des informations *endogènes* sur les composants (réception/retour de message, manipulation des politiques, modifications du contenu, des connexions, du cycle de vie...) ou *exogènes* sur le contexte d'exécution (changement de valeur, réalisation d'une condition...). Ces informations collectées peuvent aussi être primitives ou combinées par des opérateurs logiques ou chroniques. Un système d'*adaptation*, quant à lui, décrit les changements de contexte précédents à détecter et les politiques d'adaptation à appliquer. Ces politiques sont exprimées sous la forme de règles ECA, et les actions d'adaptation représentent des reconfigurations qui modifient la structure de l'application (ajout/suppression de composants, (dé-)connexions d'interfaces) ou les attributs des composants. Les logiques d'adaptations peuvent être dynamiquement attachées/détachées et cela est fait individuellement par composant en utilisant l'interface de contrôle dédiée. De plus, chaque composant à individuellement des logiques d'adaptations et les interactions de plusieurs politiques sur un même composant sont résolues par sélection, fusion ou traitement en séquence. Enfin, un langage dédié pour programmer plus facilement les reconfigurations structurelles dans des architectures *Fractal* est aussi défini (navigation, sélection d'éléments, accès aux primitives de reconfigurations de *Fractal*...). La spécification de ce langage définit des critères de consistance concernant les reconfigurations, et par exemple l'implémentation qui s'adresse au modèle *Fractal*, organise les reconfigurations selon un enchaînement d'opérations bien précis qui *i)* vérifie quelques propriétés d'intégrité structurelle relatives au modèle *Fractal* (pas de composants stoppés implicitement avec des interfaces clientes obligatoires non connectées,...) ou des contraintes d'architectures propres à chaque composant, et *ii)* journalise chaque action primitive de reconfiguration pour pouvoir les rejouer à l'envers, en cas d'échec de reconfiguration.

Le modèle de composants adaptatifs *ACEEL* [Chefrour 2005] a été proposé dans l'équipe *Paris*, pour mettre en œuvre l'adaptation dynamique en combinant des techniques de réflexion et l'utilisation de patrons de conception. Ce modèle fait le choix de se concentrer spécifiquement sur le développement de composants adaptatifs par construction, en isolant clairement la partie *adaptable* des composants. Elle définit sa propre notion de composant (peu de structuration, pas de notion de connexions d'interfaces,...), et les capacités d'adaptations sont donc très spécifiques à cette plate-forme. Dans *ACEEL*, la surveillance et l'adaptation sont surtout opérés *localement* à chaque composant, et la plate-forme permet de développer des applications adaptatives par changement d'implémentations. Pour cela, un composant n'est pas une unité d'encapsulation indivisible mais clairement formé de plusieurs parties : un *contexte* qui implémente l'interface externe et délègue les appels vers une implémentation courante, une partie qui encapsule uniquement *l'état* du composant et, plusieurs autres implémentations qui vivent dans le composant et se partagent son état. Les logiques d'adaptations sont exprimées dans des scripts séparés et spécifiées sous la forme de couples (événement, action) dont les événements sont générés par un

système de surveillance, et les actions consistent à modifier la valeur d'un attribut du composant ou à remplacer l'implantation courante du composant parmi les autres disponibles. Ces logiques sont chargées en mémoire et exploitées par un *adaptateur* qui réside au niveau méta du composant. Le système de surveillance observe diverses ressources système et est implémenté sous la forme de démons qui notifient ou peuvent être interrogés par les adaptateurs des composants, en utilisant un protocole textuel dédié. Enfin, il est à noter que la gestion de la cohérence dans *ACEEL* est faite par construction – la partie état est isolée et distincte des implémentations remplaçables – et revient surtout à isoler le composant en cours de reconfiguration des messages qu'il reçoit.

Le canevas *Dynaco* (*Dynamic Adaptation for Components* en anglais) [Buisson 2006] est proposé pour concevoir des applications à base de composants adaptables dans le contexte des grilles de calculs. Ce canevas décompose clairement les adaptations selon les quatre étapes successives *i)* d'observation (détection des changements), *ii)* de décision (choix de la stratégie), *iii)* de planification (définition des actions qui satisfont la stratégie choisie) et *iv)* d'exécution (réalisation des actions). A chacune de ces étapes, *Dynaco* associe des entités dédiées, et celles-ci sont définies indépendamment des logiques des composants métiers. Le prototype actuel de *Dynaco* est basé sur le modèle à composants *Fractal*, et les différentes entités qui implémentent les fonctionnalités d'adaptation précédentes sont implémentées sous la forme de contrôleurs *Fractal* regroupés dans un composite, hébergé dans la membrane des composants. Ces entités exposent diverses interfaces, internes ou externes à la membrane, qui permettent de définir les logiques d'adaptations (événements à surveiller, actions d'adaptations à effectuer...), et celles-ci peuvent aussi porter sur les entités de l'adaptation. Enfin, une mise en œuvre complète de ce canevas est effectuée avec le système *AFPAC* pour l'adaptation de composants parallèles (surveillance de ressources allouées et adaptation de leurs utilisations...).

3.5 Synthèse

Pour ce qui concerne les mécanismes de spécification, mis à part les langages de spécification définis pour des domaines d'application bien précis (intergiciels, multimédias,...), les langages plus généraux (*QML*, *CQML* et *CQML+*) proposent des abstractions principales pour définir successivement les concepts et les contraintes extrafonctionnelles. Ils reposent selon globalement le même schéma : *i)* la définition des concepts extrafonctionnels pertinents d'intérêts, *ii)* la quantification consiste à associer des contraintes aux concepts définis précédemment et *iii)* la mise en correspondance (*mapping* en anglais) pour lier les contraintes de qualité avec les éléments de l'application (méthodes, interfaces, composants,...). Les spécifications extrafonctionnelles consistent souvent en des définitions de niveaux de qualité, exprimés par des intervalles de valeurs valides. De plus, les concepts extrafonctionnels introduits sont souvent abstraits et avec peu de liens avec les éléments applicatifs à l'exécution, même si certains langages permettent de référencer des éléments à l'exécution. L'association des contraintes extrafonctionnelles aux éléments applicatifs est réalisé uniquement au travers des profils. Il en résulte que dans les cas les plus favorables, des structures de données sont construites de façon *adhoc* pour représenter les spécifications à l'exécution, et les mécanismes d'adaptation consistent en des modifications des profils de QdS initiaux.

Des travaux plus complets se basent sur des contrats de QdS et proposent aussi un support des contrats à l'exécution qui consiste en une infrastructure propre chargée d'interpréter et de gérer les contrats. Dans ces propositions, les contrats servent le plus souvent à effectuer de la réservation des ressources et à surveiller l'utilisation qui en est faite. De plus, les mécanismes de négociation, lorsqu'ils existent, consistent à établir le contrat qui va spécifier les niveaux requis par les applications et fournis par le système. Il est construit par mise en correspondances des profils de QdS et est ensuite échangé entre les différentes entités de l'infrastructure pour opérer la réservation effective, avec ou sans surveillance. La surveillance

des contrats reviennent donc ainsi à signaler les violations de contrats. Certains travaux spécifiques permettent aussi de spécifier des logiques d'adaptation au moment des spécifications extrafonctionnelles. Ces logiques décrivent des méthodes à exécuter lors des transitions d'états, représentées par des changements de domaine de valeurs de la QdS mesurée. D'un autre coté, les travaux qui se concentrent à la mise en œuvre de l'adaptation dynamique ne s'appuient pas sur des technologies de spécifications ou de contrats. Elles proposent diverses techniques d'adaptations qui sont exprimées par des règles ECA ou des automates, et pour ceux qui s'appuient sur des plates-formes à composants, les logiques d'adaptation reviennent à effectuer des modification d'implémentation de composants ou de reconfigurations d'attributs.

4 Négociation dynamique

Après avoir étudié les principaux travaux dans les domaines des composants contractualisés et des aspects extrafonctionnels, ce dernier chapitre de l'état de l'art présente un aperçu général de la négociation dynamique. Compte tenu de sa généralité, ce chapitre commence par fournir une vision d'ensemble des concepts, des mécanismes sous-jacents et des alternatives à la négociation dynamique. Puis, elle présente les principales propositions effectuées dans le domaine des systèmes multi-agents.

4.1 Vision d'ensemble de la négociation

4.1.1 Contexte

Lorsque des entités (humains, objets, composants,...) ont un certain niveau d'*autonomie*, elles sont dotées de leurs propres compétences et agissent souvent en vue de satisfaire leurs propres intérêts [Briot et Demazeau 2001]. Compte tenu de cette autonomie, les activités des entités autonomes, peuvent être complètement indépendantes, mais bien souvent, comme celles-ci ne sont *a priori* pas organisées, leurs activités peuvent alors tout à fait se recouvrir et leur environnement devient donc partagé. Il en résulte ainsi des interdépendances qui concernent le plus fréquemment *i)* les objectifs (buts communs, partagés, individuels mais interdépendants), *ii)* les dépendances aux ressources (ressources limitées, partagées,...) ou encore *iii)* les dépendances fonctionnelles (besoin des capacités d'autres), et qui imposent alors de fait des *interactions* pouvant consister par exemple à coordonner les actions individuelles pour l'élaboration d'un plan commun ou encore à accorder les différents points de vue pour résoudre des conflits.

Dans ce contexte, la négociation constitue une abstraction fondamentale et évoluée permettant de gérer dynamiquement les interactions, en particulier lorsque les entités dotées de leur propre intelligence et organisées de façon distribuée, cherchent à accomplir leurs buts. Ainsi, elle est usuellement utilisée pour la résolution de problèmes globaux qui tiennent compte de buts individuels, l'optimisation collective d'une solution globale par des efforts individuels ou encore la résolution des conflits engendrés par des actions interdépendantes. Toutefois, parce qu'elle représente un concept général, la négociation reste assez floue et difficile à appréhender. Elle introduit de nombreux concepts, intègre différentes dimensions, et sa mise en œuvre effective peut être faite sous diverses formes, selon différents contextes et à différents degrés de complexité. Sous une forme basique, les interactions sont mises en œuvre de façon simple par un transfert d'informations explicite entre entités (échange intentionnel de messages) et, du

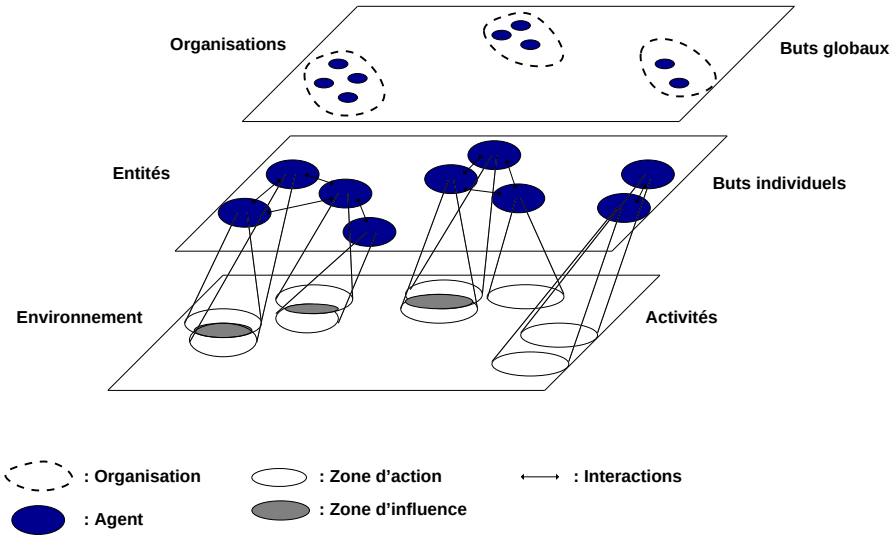


FIGURE *4.1 – Représentation des entités et de leurs interactions*

point de vue de chaque membre, elles se décomposent selon les phase de *i)* réception d'informations, *ii)* raisonnement à partir des informations acquises et *iii)* émission d'informations ou réalisation d'actions visant à modifier l'environnement ou les entités. D'autres formes d'interactions plus évoluées peuvent aussi être définies selon que les entités aient des connaissances plus étendues (capacité omnipotente, dépendance envers d'autres membres, etc.), des capacités de planifier ou de partitionner leurs actions ou encore de faire évoluer leurs raisonnements par *apprentissage* en observant les actions des autres.

4.1.2 Principales définitions

Selon une première proposition, issue des travaux de R. Davis et R. Smith concernant le *Contract-Net Protocol* [Smith 1980; Smith et Davis 1980], la négociation est vue comme «*une discussion entre des parties intéressées qui s'échangent des informations pour parvenir à un accord*». En complément de cette définition, ils présentent trois composantes de base, nécessaires pour définir une situation de négociation : a) un échange mutuel d'information, b) la capacité de chaque partie d'évaluer les informations de sa propre perspective, et c) la sélection du choix final effectué parmi les différents résultats possibles. Dans cette définition, la négociation est plutôt vue sous la forme d'un processus d'organisation qui permet la résolution des problèmes en décomposant le problème en sous-tâches et allouant ces dernières. Elle intègre cependant aussi des notions importantes de **communication** (cf. a)) et de capacités de **raisonnement** (cf. b) et c)). D. Pruitt considère la négociation comme «*un processus par lequel une décision commune est réalisée par deux ou plusieurs parties*» [Pruitt 1981], et détaille davantage les **aspects stratégiques** liés aux raisonnements des parties : «*les parties commencent d'abord par formuler leurs demandes contradictoires puis, évoluent vers un accord en faisant des concessions ou en recherchant des alternatives*». La définition de Pruitt met en avant les raisonnements internes pour converger vers un accord. Pour E. Durfee et V. Lesser, la négociation est un «*échange structuré d'informations permettant d'améliorer les accords sur des points de vue communs ou des plans d'actions*» [Durfee et Lesser 1989]. Enfin, d'une façon plus technique, N. Jennings propose de considérer la négociation comme étant un processus de recherche de solution dans l'espace des différents accords potentiels [Jennings *et al.* 2000, 2001]. Dans cette représentation, chaque proposition formulée est représentée par un point d'un espace de dimension égale au nombre de critères sur lequel porte la négociation. L'ensemble des propositions

des participants forme alors différentes régions qui évoluent et l'objectif de la négociation consiste à trouver un point acceptable appartenant aux régions des différents participants.

Toutes ces différentes définitions présentent la négociation selon plusieurs points de vue, en insistant en particulier sur les dimensions de communication et de raisonnement des participants. Plus précisément, les caractéristiques d'une négociation qui ressortent le plus souvent sont les suivantes : *i)* des échanges structurés *ii)* entre plusieurs parties *iii)* avec pour but de parvenir à un accord *iv)* en partant de points de vue contradictoires, avec la capacité pour chaque partie *v)* d'évaluer les informations de sa propre perspective et *vi)* de faire des concessions ou de rechercher des solutions alternatives.

4.1.3 Aperçu des composantes principales

En dépit de la variété des domaines dans lesquels la négociation s'applique et des différentes approches possibles, la très grande majorité des négociations automatisées définissent usuellement certaines grandes dimensions. Tout d'abord, la négociation doit préciser ce sur *quoi* elle porte. Ces **objets de la négociation** (appelés aussi termes ou critères négociés) varient en fonction des domaines et des applications. Par la suite, la négociation met en jeu différents participants (appelés aussi parties) qui interviennent au cours de la négociation. Ces différentes **parties négociantes** communiquent et s'échangent des informations tout le long du processus de négociation pour que celui-ci avance. L'ensemble de ces échanges ainsi que le déroulement général de la négociation sont gouvernés par un ensemble de règles, au travers des **protocoles de négociation**. Enfin, les parties négociantes sont équipées de **capacités internes de décision** qui leur permet de raisonner et d'agir en accord avec le protocole de négociation et de façon à atteindre les objectifs.

Objectif de la négociation

D'une façon générale, l'objectif de la négociation est de résoudre des conflits ou de coordonner les actions entre entités capables de raisonner et de communiquer. En présence d'une forme de conflits, et c'est en particulier le cas lorsque les entités ont des objectifs opposés ou qu'ils sont en conflits sur des ressources, la négociation devient un moyen pour la résolution des problèmes. Dans un environnement davantage coopératif, lorsque différentes entités veulent coordonner leurs tâches, la négociation contribue alors à obtenir une plus grande coordination entre les entités. Elle peut ainsi intervenir pendant la décomposition d'un problème initial en sous-problèmes, pendant l'exécution individuelle des sous-problèmes ou encore après l'exécution des plans individuels au moment de recomposer la solution du problème initial. De ce point de vue, la négociation est ainsi présentée comme un processus permettant de décomposer et de recomposer d'une manière cohérente, et d'orienter les comportements individuels vers des buts communs. La négociation devient ainsi un moyen, qui se justifie notamment bien dans les environnement distribués et permet la résolution de deux classes de problèmes [Martial 1990] : la planification dirigée par les tâches et la coordination de plans. Dans la planification dirigée par les tâches, il existe un problème initial qui est décomposé en plusieurs sous-problèmes répartis entre plusieurs entités (décomposition *top-down* du problème dans un modèle d'organisation hiérarchique), et l'objectif est de trouver la *bonne* décomposition. Dans un problème de coordination de plans, les plans d'agents existent déjà et il s'agit alors de concilier les plans individuels avant leur exécution.

Termes négociés

Compte tenu des objectifs généraux de la négociation, l'éventail des termes sur lesquels elle porte est aussi très large. Les termes négociés peuvent consister en un seul terme unique ou alors en des termes plus complexes, éventuellement liés. Dans l'exemple classique des ventes aux enchères, la négociation

porte souvent sur les prix des objets en cours mais d'autres critères peuvent aussi entrer en compte comme les conditions de garantie et de livraisons ou encore les pénalités. Les différents termes de la négociation restent donc complètement ouverts et sont à définir par le domaine dans lequel s'applique la négociation. Dans les systèmes multi-agents, l'étendue des termes sur lesquels peut porter la négociation est extrêmement variée. Ils peuvent ainsi porter sur les objets du domaine d'application, les objets de contrôle de la négociation et éventuellement certains autres paramètres de qualité. Les termes négociés ne sont pas réduits à des domaines particuliers et la richesse des concepts introduits permet potentiellement d'envisager la négociation de termes quelconques du moment que des mécanismes permettant de décrire et de raisonner sur ces termes soient définis. Néanmoins, il est possible de regrouper les termes selon qu'ils représentent :

- des objets définis par les domaines d'application de la négociation et qui sur lesquels portent la négociation. Ces objets peuvent être plus ou moins complexes. Il n'y a encore une fois aucune restriction sur la nature des objets, et par exemple, la négociation peut porter sur des termes relativement simples comme des intervalles de temps lorsqu'il s'agit d'organiser des rendez-vous, des prix de produits dans les systèmes de commerces ou de ventes aux enchères, mais aussi sur des termes plus complexes comme le partage de ressources ou encore des ensembles de termes liés (prix de produits liés à des conditions de garanties et de livraisons ou à des pénalités,...). Sous une forme plus élaborée, la négociation est aussi employée pour le partage de ressources [Cammarata *et al.* 1983], la gestion du trafic aérien [Wollkind *et al.* 2004] ou encore la gestion d'informations distribuées [Jennings *et al.* 1996] ;
- des éléments qui décrivent le déroulement même de la négociation. Dans cette seconde catégorie, il s'agit de négocier sur la manière dont se déroule le processus de négociation lui-même. En effet, comme l'approche par négociation est ouverte, il est aussi possible de structurer une négociation selon plusieurs niveaux pour pouvoir négocier, sur un premier niveau, les objets directs des conflits, et sur un second niveau, la manière de conduire la résolution elle-même (proposition de nouveaux critères, réduction à un sous-problème moins contraints,...).

Parties négociantes

Les parties négociantes ne sont pas imposées et sont à définir par les applications. Les parties peuvent être organisées de plusieurs façons différentes (négociation bilatérale, ou multi-latérale), et chaque partie a souvent un rôle bien précis à tenir au cours de la négociation. Dans les systèmes multi-agents (SMA), les parties sont les agents, et bien que les modèles de négociation dans les SMA restent ouverts, en pratique les négociations sont souvent bilatérales (une unique négociation bilatérale isolée ou plusieurs négociations bilatérales indépendantes). Dans les travaux portant sur la gestion de la Qualité de Service, les acteurs ne sont pas définis. Les parties sont aussi souvent bilatérales, car liées par une relation de type client-fournisseur (interactions entre clients-fournisseurs pour l'utilisation et la fourniture de services, connexions entre composants clients/serveurs pour l'appel et la réalisation de services). De plus, comme l'objectif principal est surtout de gérer de façon efficace les paramètres de qualité, les rôles ne sont pas très clairement explicités.

Mécanismes généraux

De façon générale, la négociation est un processus d'échanges d'informations entre les différentes parties impliquées. C'est un processus itératif qui repose sur des communications structurées et dans lequel les différentes parties dans la négociation agissent selon des stratégies d'échanges permettant de résoudre le problème. Ainsi, au fur et à mesure des échanges successifs, les parties recueillent des informations sur l'état courant de la négociation et éventuellement sur les autres parties. Ces informations

leur permet alors d'élaborer successivement des nouvelles propositions jusqu'à ce qu'un compromis soit trouvé ou qu'un conflit majeur apparaisse. Par la suite, plusieurs formes de négociation peuvent être mises en œuvre selon *i)* la structure et les rôles des parties négociantes (cardinalité des participants, rôles, éventuelles parties tierces), *ii)* les communications entre les différentes parties (description des objets négociés, langages de négociation), *iii)* le processus général (différents états de la négociation et leur transition) ou encore *iv)* les raisonnements des parties négociantes leur permettant de guider leur comportement et d'atteindre leurs objectifs (modèle de prise de décision, actions possibles).

Pour décrire plus précisément ces mécanismes de négociations, il faut s'intéresser aux mises en œuvre réalisées dans des domaines plus spécifiques. La section qui suit décrira dans le détail la modélisation des comportements intelligents lors des négociations et les mécanismes de négociation définis.

4.1.4 Approches alternatives

D'autres approches alternatives à la négociation ont été proposées. La suite présente brièvement ces propositions selon qu'elles s'orientent davantage vers des procédés de coordination, ou qu'elles constituent des techniques de résolution qui se focalisent sur l'obtention de la solution.

Approches orientées coordination

La coordination est présentée comme étant un procédé qui permet à des entités d'évoluer d'une façon cohérente sans que leurs activités ne rentrent en conflit. Il existe plusieurs techniques de coordination : les structures organisationnelles dans lesquelles la coordination s'appuie fortement sur les rôles et les relations des membres, la planification qui consiste à composer de façon adéquate des plans individuels afin d'éviter les conflits futurs, et la contractualisation entre un gestionnaire et des contractants pour l'allocation des tâches et des ressources.

Structures organisationnelles Il s'agit à *a priori* de la technique de coordination la plus simple. Ici, toute la coordination repose sur l'existence au, sein de l'organisation, d'une représentation des rôles des membres, de leurs relations et de leurs capacités. Cette forte explicitation des rôles et des relations fournit alors un cadre qui régit les activités des membres [Durfee *et al.* 1987]. Parmi les différentes organisations possibles, on trouve les habituelles organisations hiérarchiques incluant les relations usuelles *maître-esclaves* ou *serveur-clients*, dans lequel un membre central exerce son autorité sur le reste des membres et contrôle leurs actions et leurs plans. Les membres-esclaves, quant à eux, peuvent ou pas communiquer entre eux, mais doivent, à tout prix, rendre compte de leurs actions au membre-maître. Dans ce cas, l'autonomie (d'actions, de décisions) des esclaves est très faible et le contrôle exercé par le maître est total. Il est à noter que la plupart des structures organisationnelles sont bâties sur une organisation hiérarchique qui existe de fait, et qui permet donc de centraliser le contrôle et l'affectation des actions. En revanche, cette centralisation limite considérablement bon nombre d'avantages de la distribution (peu de parallélisme, goulot d'étranglement,...) et va un peu à l'encontre de l'approche par intelligence distribuée [Durfee *et al.* 1989]. De plus, cela suppose aussi qu'un membre spécifique dispose de la connaissance globale du problème lui conférant ainsi, pour la résolution du problème, le droit de contrôle sur ses compatriotes.

Contractualisation Une technique classique de coordination pour la répartition des tâches et pour l'allocation des ressources entre entités est proposée par la famille des protocoles du *Contract-Net Protocol* [Smith 1980; Davis et Smith 1988]. Cette approche préconise une structuration des entités en organisation de marché telle que chaque entité peut avoir :

- soit le rôle de *gestionnaire* qui divise un problème global en plusieurs sous-problèmes, cherche à affecter ces sous-problèmes à plusieurs contractants et supervise leurs résolutions ;
- soit le rôle de *contractant* qui accomplit et résout un sous-problème. De façon récursive, un contractant peut aussi à son tour devenir gestionnaire et re-décomposer son problème en divers sous-problèmes s'il ne possède pas les capacités pour le résoudre seul.

C'est une technique complètement distribuée dans laquelle chaque intervenant peut à la fois être gestionnaire et contractant. De plus, en comparaison, à la technique basée sur une organisation hiérarchique, la réalisation de chaque sous-problème est à la charge de chaque contractant et dépend de ses propres capacités. Le gestionnaire effectue simplement la décomposition du problème en sous-problèmes, la répartition des sous-problèmes et la collecte des solutions de chaque sous-problème. En revanche ses principales limitations concernent le fait qu'une situation de coordination initiale est supposée exister et que les contractants sont assez bénévoles et non-antagonistes. Cette technique ne permet ainsi pas de détecter, ni de résoudre des conflits. De plus, le coût des communications semble être assez élevé, en particulier induit au moment de la recherche des contractants. Compte tenu des notions d'organisation hiérarchique et de contractants introduits par le *Contract-Net Protocol*, une étude plus complète de ce protocole et de son utilisation dans les systèmes multi-agents est effectuée dans la section suivante 4.2.

Planification Dans le but d'éviter toute activité et interaction conflictuelle, une autre forme de coordination consiste à planifier les activités des différentes entités. Ainsi, avant d'entreprendre toute action, les entités s'entendent pour construire un plan global qui détaille le découpage des plans individuels de chacun (leurs actions futures et leurs interactions), ainsi que les éventuelles étapes de recoupement pour la re-planification ou une autre réorganisation des plans individuels [Edmund H. Durfee 1987; Georgeff 1988a]. Il existe deux formes de planification, centralisée et décentralisée.

Dans une planification centralisée [Georgeff 1988a,b; Cammarata *et al.* 1983], il existe usuellement un coordinateur général qui, après réception des plans individuels, les analyse pour identifier les conflits potentiels ainsi que les interactions inconsistantes (ex. ressources requises insuffisantes). Par la suite, le coordinateur général tente de modifier les plans individuels et de les organiser d'une façon cohérente afin d'éliminer l'ensemble des conflits exhibés à l'étape précédente. Dans la forme décentralisée [Durfee et Lesser 1987; Corkill 1979], le coordinateur général n'existe plus et chaque entité doit avoir une représentation des plans individuels des autres. Ainsi, les agents doivent se communiquer et s'échanger leurs plans individuels afin de les modifier ou de les mettre à jour pour résoudre les conflits.

Généralement, les techniques de planification sont fortement consommatrices de ressources car des échanges d'informations importants sont réalisés. La forme centralisée reprend les mêmes limitations que les techniques de coordination de type *maître-esclaves* alors que la mise en œuvre de la planification décentralisée est beaucoup plus complexe, car chaque entité ne dispose justement que d'un point de vue local du problème.

Approches orientées résolution

Vote L'approche par vote, stratégie usuellement utilisée dans les sociétés humaines, propose des processus de décision basé sur le choix d'une solution parmi un ensemble de solution prédéterminé : un ensemble de solutions existe ; chaque membre effectue son vote et la solution finale est celle qui recueille l'unanimité ou la majorité des voix. En comparaison de la négociation, l'approche basée sur le vote reste exclusivement employée pour résoudre des questions simples et n'est pas applicable dans des situations très complexes car le processus de déroulement est fermé et très ciblé, la solution finale fait partie d'un ensemble de solutions connu dès le départ, et les conflits ne sont pas explicités et ne peuvent pas être modifiés. En revanche, elle pourrait très bien être employée comme une étape postérieure à la négociation pour permettre de choisir entre des solutions globalement équivalentes.

Théorie des jeux Un jeu est un modèle formel de diverses situations d'interactions : plusieurs joueurs rationnels interviennent ; ils ont des individuellement des préférences ; disposent de certaines informations ; et effectuent par le biais de stratégies des actions qui ont des influences sur les autres joueurs. La théorie des jeux constitue donc une théorie de la décision rationnelle d'individus stratégiquement interdépendants. Il s'agit de modéliser le processus de décision interne des joueurs dans une situation où ils sont égocentrés [Osborne et Rubinstein 1994; Osborne 2004] ou coopératifs et d'étudier comment les actions des uns influencent celles des autres [Luce et Raiffa 1957]. Le calcul de l'action optimale n'est pas forcément une opération triviale et peut se révéler coûteux, voire impossible. De nombreux travaux ont abordé la théorie des jeux par des modélisations mathématiques et économiques du problème. Toutefois, en dépit de l'élégance des solutions mathématiques proposées, de nombreuses hypothèses limitent fortement son application à certains problèmes réalistes. Ainsi, les négociations de la vie courante, sont souvent conduites sous une certaine incertitude, ils portent davantage sur plusieurs termes corrélés que sur le simple critère de gain, les connaissances et stratégies de joueurs ne sont pas communes mais plutôt privées et les individus ne sont pas omniscients. Une part importante des travaux de la négociation qui se basent sur la théorie des jeux ont été appliqués dans les systèmes multi-agents par Rosenschein et Zlotkin [Rosenschein et Zlotkin 1994] dans leur étude des mécanismes de coordination entre agents rationnels.

Techniques de résolution de contraintes Les problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem* en anglais), qui apparaissent souvent dans les problèmes d'allocation de ressources et planification, consistent à chercher une solution qui satisfasse un système de contraintes, par le calcul des valeurs de variables adéquates. La résolution de ces problèmes est généralement combinatoire et donc des *raisonnements* et des *heuristiques* particuliers sont souvent introduits pour permettre de guider la recherche vers les bonnes combinaisons et ainsi réduire l'examen des combinaisons possibles. Des algorithmes généraux mais moins efficaces peuvent être appliqués directement et dans certains cas, en fonction de la formes des contraintes ou des variables, des algorithmes spécialisés mais plus effeicaces peuvent être définis. Au lieu de résoudre les CSP avec les techniques classiques, d'autres approches ont aussi proposé de distribuer ce problème de résolution et de faire appel à des techniques issues traditionnellement de l'Intelligence Artificielle. Ainsi, dans un CSP distribué [Fernandez *et al.* 2002; Hamadi 1999], les variables sont distribuées entre différentes entités et les relations entre ces entités sont déterminées par le système de contraintes. Dès lors, l'algorithme de résolution est réalisé par chaque entité sur les variables qu'il a en charge et les agents communiquent et s'échangent des informations pour valider ou réfuter les valeurs affectées aux variables par les uns et les autres. Ces techniques sont basées *in fine* sur la modélisation et la résolution d'un système de contraintes sans qu'il n'y ait réellement de processus d'interactions.

Evaluation

Nous avons brièvement présenté une liste des principales approches de résolution de conflits et de coordination, alternatives à la négociation. Les structures organisationnelles, la contractualisation et la planification abordent particulièrement le problème sous l'angle de la coordination alors que les approches par vote, par la théorie des jeux ou encore la résolution de contraintes proposent davantage des techniques de résolution spécifiques et ciblés à certains types de problèmes bien précis.

En guise de synthèse, le tableau 4.1 récapitule ces différentes techniques selon les critères suivants :

- *Rôles et relations :* les rôles des entités et leurs relations peuvent être explicites. Par exemple, ces rôles sont prédéfinis dans les structures hiérarchiques ou dans les organisations de marché (maître-esclaves, fournisseurs-consommateurs,...).
- *Type de communication :* il peut y avoir un processus de communication explicite permettant l'échange d'informations entre les différentes entités.

– *Connaissances :* les connaissances peuvent être centralisées en une seule entité de contrôle, ou alors réparties dans chacune des différentes entités.
– *Solution :* la solution peut être soit élaborée dynamiquement à partir des interactions entre les entités, soit choisie à partir d'un ensemble donné ou alors soit calculée.
– *Décision :* la décision peut être collective lorsqu'elle est le fruit d'échanges successifs, centralisée si une entité détient un rôle prépondérant ou alors individuelle lorsque les entités sont égocentrés et que chacun tente de réaliser ses buts individuels.

	Rôles et relations	**Type de communication**	**Connaissances**	**Solution**	**Décision**
Structures organisationnelles	explicites	distribuée, ouverte	réparties	*indéfinie*	centralisée ou collective
Contractualisation	explicites	distribuée, de type appel-d'offres	réparties	élaborée dynamiquement	centralisée ou collective
Planification	*indéfinis*	distribuée	centralisée ou répartie	élaborée dynamiquement	centralisée ou collective
Vote	*indéfinis*	simple	réparties et limitées	choisie	à la majorité
Théorie des jeux	*indéfinis*	*indéfini*	réparties et partagées	calculée	individuelle
Résolution de contraintes	*indéfinis*	*indéfini*	centralisée ou répartie	calculée	centralisée ou collective

TABLE *4.1 – Synthèse des approches alternatives à la négociation.*

Par ailleurs, en remettant en perspective chacune des techniques présentées par rapport à la caractérisation de la négociation présentée en section 4.1.2 comme étant constitué par *«des échanges structurés entre plusieurs parties avec pour but de parvenir à un accord (en partant de points de vue contradictoires), avec la capacité pour chaque partie d'évaluer les informations de sa propre perspective et de faire des concessions ou de rechercher des alternatives»*, on constate que chacune des techniques aborde une partie de ces caractéristiques :

– les structures organisationnelles insistent sur les rôles et les relations des parties. En particulier, les structures hiérarchiques laissent entrevoir une forme de négociation *autoritaire* dans laquelle une entité contrôle et influence les autres. En revanche, le processus de négociation n'est pas du tout abordé ici ;
– la contractualisation s'attache justement, et exclusivement, à cet aspect. En se plaçant dans le contexte très précis des organisations de marché, elle s'attache à décrire les mécanismes d'échanges pour traiter le problème particulier de l'allocation de tâches et de ressources. En plus d'être spécifique à une certaine classe de problème, la contractualisation ne s'attarde pas sur les capacités internes de raisonnements des participants, gestionnaire ou contractants ;
– la planification se place aussi dans un contexte bien précis où il s'agit d'organiser les actions des uns et des autres avant leur exécution de façon à ce que l'ensemble des activités puisse se dérouler d'une façon cohérente sans conflit. Les échanges et les raisonnements des différents intervenants ne sont pas abordés ;
– le vote constitue certainement la technique la plus rompue et la plus spécifique parmi celles présentées. Il constitue bien un processus permettant de parvenir à un accord, cependant, les échanges sont simplistes, et le processus n'est guère évolué (choix des solutions parmi un ensemble donné, raisonnements simples justifié par le fait que le vote doit porter sur des termes simples, décision à

la majorité) ;
– la théorie des jeux s'intéresse essentiellement au processus de décision et donc peut intervenir pour concevoir les capacités internes de participants rationnels. Toutefois, une étude approfondie serait nécessaire afin d'analyser si certains résultats de cette théorie seraient transposables à la forme de négociation qui nous intéresse (adéquation des modèles de raisonnements, hypothèses d'application,...) ;
– enfin, les techniques de résolution de contraintes constituent à elle seules des techniques à part entière dont l'objectif est de trouver une solution à un système de contraintes. L'hypothèse implicite consiste donc à représenter les conflits sous une forme équationnelle et l'utilisation de telles techniques dans une approche par négociation serait conditionnée par l'applicabilité des hypothèses d'utilisation. En revanche, elles pourraient offrir un moyen permettant de modéliser ou d'orienter les capacités de raisonnement interne ;

	Rôles et relations	**Echanges structurées**	**Obtention d'un accord**	**Capacités internes**
Structures organisationnelles	⋆⋆⋆	⋆	⋆	⋆
Contractualisation	⋆⋆	⋆⋆⋆	⋆⋆⋆	⋆
Planification	⋆	⋆	⋆⋆⋆	⋆
Vote	⋆	⋆	⋆	⋆
Théorie des jeux	⋆	⋆	⋆⋆⋆	⋆⋆
Résolution de contraintes	⋆	⋆	⋆⋆⋆	⋆⋆

⋆⋆⋆ : point fort
⋆⋆ : notion existante mais peu aboutie ou spécifique
⋆ : peu évoqué

TABLE *4.2 – Évaluation des techniques vis-à-vis de la négociation.*

Les techniques qui viennent d'être présentées s'attardent sur certaines caractéristiques bien spécifiques de la *négociation*. Ainsi, il apparaît que la négociation doit être vue comme une approche réellement générale qui englobe en particulier les approches que nous venons d'évoquer. La négociation permet donc d'abstraire les techniques classiques de résolution de conflits et de coordination d'activités par le biais de certaines briques de base.

4.2 Négociation dans les systèmes multi-agents

Dans cette section, nous nous intéressons particulièrement à la négociation dans les systèmes multi-agents (SMA). Il s'agit, en effet, d'une discipline dans laquelle le concept de négociation occupe une place fondamentale, et de nombreux travaux y sont proposés. Après avoir présenté dans une première sous-section, le cadre général des agents et des systèmes à agents, nous introduirons le contexte de la négociation et présenterons ses motivations. Puis, nous décrirons les principes généraux des mécanismes de négociation dans les systèmes à agents. Enfin, nous présenterons les systèmes de négociation significatifs dans ce domaine.

4.2.1 Cadre général

Systèmes Multi-Agents

Dans les systèmes multi-agents, les agents sont considérés comme des entités de base *intelligentes* qui ont diverses caractéristiques [Ferber 1995; Franklin et Graesser 1997; Huhns et Singh 1998; Jennings *et al.* 1998.]. Ils sont *i) autonomes* (l'agent contrôle ses actions et agit sans intervention assistée), *ii) situés* (l'agent est capable d'agir sur l'environnement à partir de messages reçus ou de perceptions sensorielles), *iii) proactifs* (l'agent est capable de prendre des initiatives et n'agit plus seulement en tant qu'exécutant d'une action dictée), *iv) capables de percevoir leur environnement* et d'élaborer une réponse ou de réaliser une action, et enfin *v) capables d'interagir en société* de façon à interagir avec d'autres agents dans le but de compléter leurs actions ou aider les autres à l'accomplissement des leurs.

Munis de ces capacités individuelles, les agents font alors partie d'un environnement dit multi-agents, dans lequel ils sont amenés à collaborer en s'appuyant sur leurs compétences et leurs connaissances individuelles. Par extension à la notion d'agent, la thématique des systèmes multi-agents se focalise sur l'étude des comportements collectifs et sur la répartition de l'intelligence sur des agents plus ou moins autonomes, capables de s'organiser et d'interagir pour résoudre des problèmes. Cette vision devient d'autant plus pertinente dans le contexte actuel où la très grande majorité des systèmes sont de fait distribués et deviennent de plus en plus ouverts. Un système multi-agents peut ainsi être vu comme « *un réseau d'agents faiblement interconnectés qui travaillent ensemble dans le but de résoudre des problèmes qui ne peuvent pas être résolus individuellement par les agents* » [O'Hare et Jennings 1996].

Ainsi, les systèmes multi-agents sont généralement définis par les caractéristiques suivantes [Briot et Demazeau 2001]. Chaque agent a des informations ou des capacités limitées et n'a aucun contrôle global du système multi-agent. De plus, les données sont souvent décentralisées et les échanges asynchrones. Par la suite, compte tenu de ces qualités, les systèmes à agents de deviennent des candidats aux qualités idéales lorsqu'il s'agit :

- de fournir une solution décentralisée pour les problèmes où l'expertise est répartie et qui ne pourraient pas l'être de façon centralisée (ressources limitées, risques inhérents aux systèmes centralisés,...) ;
- de représenter une solution naturelle pour les nombreux problèmes qui font intervenir des éléments distribuées et autonomes comme, par exemple, les problèmes de contrôle de traffic aérien [Cammarata *et al.* 1983] ;
- d'améliorer la vitesse de résolution (résolutions parallélisées à condition de maintenir des communications minimales), la fiabilité (la distribution minimise l'impact de la panne d'un agent), l'extensibilité (facilité à changer les agents qui interviennent) ;
- de favoriser l'interconnexion et la coopération entre de multiples systèmes distribués et patrimoniaux ;
- d'offrir un support conceptuel simple et claire et de tolérer un certain seuil d'incertitude sans entraver le processus de résolution.

De plus, du point de vue du processus de résolution, les systèmes multi-agents présentent l'avantage de combiner à la fois les qualités des techniques de résolution distribuée (parallélisation, redondance) et aussi celles des méthodes d'IA surtout de par l'intégration de schémas d'interaction sophistiqués comme la coopération, la coordination et la négociation. Ainsi, pour pouvoir exploiter pleinement ces systèmes, le défi majeur consiste à résoudre les problèmes inhérents à la construction des systèmes multi-agents et des mécanismes d'interactions.

Contexte et buts de la négociation

Dans les systèmes à agents, la négociation est employée en raison de la forte autonomie des agents et est employée à la fois lorsque les agents sont de nature coopérative ou égocentrée. Les buts généraux de la négociation dans les SMA consistent à faire de l'allocation de tâches ou de ressources, à détecter ou à résoudre des buts conflictuels, à modifier l'organisation du système à agents pour en garantir sa cohérence ou encore à modifier les plans des agents. Pour cela, ils doivent communiquer entre eux, par exemple en faisant des propositions et des contre-propositions de façon à modifier le comportement des autres agents. Le processus de négociation peut se présenter sous des formes diverses et variées telles que les négociations par enchères, les protocoles de la même famille que le *Contract-Net Protocol*, ou encore toutes celles qui introduisent des capacités plus évoluées d'apprentissage, d'argumentation et de croyance.

Les recherches en négociation dans les SMA se divisent en plusieurs catégories [Muller 1996]. Les recherches sur les langages de négociation analysent la syntaxe et la sémantique des messages échangés au cours d'une négociation (primitives de négociation, description des messages échangés) et leurs usages dans les protocoles (réponses autorisées, états valides). D'autres s'attardent plus sur les raisonnements des agents au cours de la négociation (évaluation et comparaison des propositions) et leurs modélisations (fonctions d'évaluation et de proposition, préférences). Enfin, une dernière catégorie étudie les processus de négociation et leur déroulement global au niveau des agents (algorithme de négociation) et au niveau du système à agents (propriétés d'équité ou d'optimalité, répartition des connaissances).

4.2.2 Principes

Termes négociés

Dans les systèmes multi-agents, le termes de la négociation sont principalement déterminés par les domaines d'applications. Ils peuvent être des objets relativement simples comme par exemple, des intervalles de temps pour des rendez-vous, des prix de produits dans les commerces électronique ou les ventes aux enchères. Mais ils peuvent aussi représenter des termes plus complexes, éventuellement liés comme par exemple, des ressources à partager, une décision commune à trouver, des objectifs ou des niveaux de qualité de service dans les transactions électroniques. Ces différents termes restent complètement ouverts et sont à définir par le domaine dans lequel s'applique la négociation. En dehors des utilisations classiques, des termes plus abstraits peuvent aussi être négociées. En particulier dans le domaine des systèmes multi-agents, il est possible de négocier les relations entre les agents (actions ou plans des entités pour leur coordination, réorganisation de la société d'agents, établissement de liens privilégiés entre agents, création d'un réseau d'interconnexion d'agents,...) [Chevrier 1992].

Protocoles et langages de communication

Communication Généralement, les communications sont définies par le contenu des messages, le mode de transmission et ce qui les motive. Les messages contiennent généralement des informations (requêtes, propositions, offres, connaissances, arguments) et des méta-informations (délai de réponse, type de négociation). De plus, dans le système, les communications peuvent se dérouler selon un mode direct ou indirect. Dans une communication directe, les agents échangent volontairement des messages en direction d'un ou d'un groupe d'agents. De plus, la communication peut se réaliser :

- soit par partage d'informations, et dans ce cas il existe une structure partagée (tableau noir) dans laquelle les agents déposent et retirent des informations,
- soit par envoi de messages selon les différents modes usuels (point-à-point, diffusion,...).

Dans une communication indirecte, il n'y pas d'échanges explicites. Des signaux (ou des traces) sont laissés sur l'environnement par des agents, par exemple suite à des actions qui modifient l'environnement, puis ces signaux sont propagés par l'environnement et perçus par d'autres agents. Il s'agit d'un mode de communication limité (perte d'efficacité car les interlocuteurs directs ne sont pas connus), mais qui peut être intéressant dans le cas où l'on souhaite, non pas réaliser des tâches précises mais plutôt des opérations grossières sur un ensemble d'agents (récepteurs) sans avoir à déterminer précisément leurs rôles. Par ailleurs, les communications peuvent aussi être soit synchrones, soit asynchrones. Dans les systèmes multi-agents, les communications synchrones sont réalisées par des appels de fonction. Les fonctions sont bloquantes pour l'appelant qui attend la réponse. Les communications asynchrones sont réalisées soit par écriture sur un tableau noir, soit par un système de messages de type boîte aux lettres. Ainsi, les agents peuvent vouloir communiquer pour prendre des initiatives, car ils se sont engagés à le faire mais aussi en réponse à la perception de changements.

Les modélisations de la communication entre agents s'appuient aussi sur la théorie des actes de langages, développée par les linguistes et qui vise à s'inspirer des dialogues entre humains. Très brièvement, dans cette théorie, un énoncé sert avant tout à produire des effets sur ses destinataires. Ainsi, les énoncés sont liés aux actes et leur donne un poids plus ou moins important. Un énoncé peut ainsi être décomposé selon trois composantes [Austin 1962]. Les actes locutoires (sujet de la conversation), illocutoires (intentions du locuteur), et perlocutoires (effets produits ou espérés). Une typologie des actes de langages a été introduite parmi lesquels les assertifs pour affirmer quelque chose sur le monde, les directifs pour donner des directives à l'allocutaire, les interrogatifs, les exercitifs pour demander d'accomplir une action, les promissifs pour s'engager à accomplir certains actes, les expressifs pour donner des indications concernant son propre état.

Langages de communication Les actes de langages ont été utilisés dans de nombreux langages pour faciliter les communications entre agents. Les langages de communication permettent aux agents d'échanger des informations en particulier pour la négociation. Parmi les plus connus, *KQML* et *ACL* considèrent les énoncés comme des actes qui visent à accomplir ou faire accomplir quelque chose aux agents allocutaires. *KQML* (*Knowledge Query Manipulation Language* en anglais) [Finin *et al.* 1993; Labrou et Finin 1997], issu des travaux de la DARPA [DARPA Knowledge sharing effort (Web Site)] propose de définir un standard de communication dans les systèmes multi-agents. Dans *KQML*, un message est défini par un identifiant (un codage du temps), l'agent émetteur du message, une liste d'agents destinataires, et le corps du message comprenant un contenu et une force illocutoire (demander la réalisation, affirmer que, répondre que,...). De plus, trois couches conceptuelles sont introduits (couche communication, couche des message et contenu du message).

Bien qu'il soit utilisé depuis plusieurs années, en particulier dans les milieux académiques, les limites principales de *KQML* proviennent de la définition vague de nombreux performatifs (ambiguïté, incohérence,...). De plus, certains performatifs (`broker`, `stream-all`), oblige les agents-expéditeurs, d'avoir, au moment de la construction des messages, une idée de la manière dont les agents-destinataires devront re-rerouter les messages. Ces limites sont abordées dans [Cohen et Levesque 1995].

Plus récemment, la FIPA (*Foundation for Intelligent Physical Agents* en anglais) [FIPA Organization (Web site)] a définit le langage *ACL* (*Agent Communication Language* en anglais) [FIPA TC Communication 2002a]. Le but principal de la FIPA est d'offrir un ensemble de spécifications qui permet de favoriser l'interopérabilité entre les agents et les systèmes multi-agents. Le langage *ACL* est aussi basé sur la théorie des actes de langages et a une syntaxe similaire à *KQML*. Toutefois, au lieu de définir des performatifs, *ACL* définit des actes de communications associées à diverses finalités (transmettre/demander des informations, distribuer des tâches, gérer un problème,...).

Les langages *KQML* et *ACL* sont grossièrement similaires. D'un point de vue pratique, *KQML*

commence à faire ses preuves mais la sémantique des actes reste peu formalisée et utilisable potentiellement dans n'importe quel contexte de communications entre agents. Le langage *ACL* a, quant à lui, été conçu directement avec une sémantique rigoureuse et commence juste a être utilisé dans des applications réelles de taille relativement importante. L'évaluation de ces deux langages est encore en cours et, les défauts révélées et les améliorations apportées tendent de plus en plus à réduire leurs différences [Wagner 1997]. Toutefois, leur utilisation reste malgré tout limitée en raison de leur complexité. La validation réelle de ces langages mais aussi les promesses d'interopérabilité dans un contexte de systèmes à agents ouverts doivent encore être développées. Un autre inconvénient de, *KQML* et d'*ACL* aussi souvent évoqué, réside dans leur approche un peu trop *académique*. Ainsi, dans certaines applications et situations de négociation, leur utilisation n'est pas justifiée et les avantages qu'ils procurent ne sont pas évidents. En pratique, il semble même que très peu d'applications réelles y ont recours.

Protocoles de négociation Les protocoles de négociation représentent un deuxième aspect majeur considéré dans les recherches portant sur la négociation automatisée. Traditionnellement, les protocoles de négociation définissent un ensemble de règles qui permettent de guider les interactions entre les agents au cours de la négociation. La fonction de base des protocoles de négociation est de permettre aux agents de communiquer et d'interagir au cours de la négociation sans qu'ils aient à planifier explicitement chacune de leur action. Ils spécifient généralement, *i)* les états de la négociation (ex. phase des offres, états de terminaison de la négociation), *ii)* les événements qui provoquent les changements d'états de la négociation (délai de réponse écoulé, proposition valide) ainsi que *iii)* les actions valides des parties négociantes dans chacune des étapes de la négociation (messages autorisés, parties interrogées). Pour la spécification des protocoles de négociation, et plus généralement des protocoles de communication, différents formalismes peuvent être employés pour une représentation expressive des interactions (graphes de raisonnement, réseaux de Pétri, langages de description formels, etc.). Toutefois, les plus classiques sont basés sur un graphe état-transition dans lequel des noeuds qui représentent l'état d'un agent sont reliés par des arcs qui représentent l'émission ou la réception d'un acte de langage. Ces formalismes s'attachent à la spécification des règles d'interaction, mais l'intérêt vient surtout du fait que comme chaque agent possède une copie du protocole, la représentation des agents peut être directement traduite dans une implémentation.

Parmi les protocoles de négociation existants, la plus grande famille est représentée par les protocoles dits *d'allocation de tâches par réseau contractuel* issus du *Contract-Net Protocol* introduit par Smith [Smith 1980] pour la résolution de problèmes distribués. Le *Contract-Net Protocol* (CNP) est un protocole général qui se focalise principalement sur la coordination des agents coopératifs, ayant les mêmes buts et qui partagent et résolvent un problème plus complexe. Un agent demande de l'aide car il n'a pas la capacité de traiter un problème donné à lui tout seul. Pour cela, il tente d'abord de partitionner la tâche initiale en sous-tâches qu'il tente ensuite d'affecter aux agents ayant les capacités appropriées à les gérer. Ainsi, dans ce protocole, la négociation se résume à l'allocation de tâches en fonction de la qualification des agents. Ce protocole établit en fait un procédé organisationnel en définissant : *(i)* un gestionnaire qui cherche divise une tâche en plusieurs sous-tâches et cherche à les répartir, *(ii)* des contractants qui répondent à l'appel d'offres et accomplissent les sous-tâches et enfin, *(iii)* des contrats qui permettent de répartir et de coordonner les sous-tâches de chaque agent pour résoudre le problème global. Le gestionnaire décompose le problème global en sous-tâches et initie la négociation en annonçant chaque sous-tâche sur le réseau. Les agents qui ont les capacités (ressources appropriées, expertise) pour réaliser la sous-tâche participent à la négociation en envoyant au gestionnaire leurs propositions. Dès lors, le gestionnaire rassemble les propositions reçues et alloue la tâche au meilleur. Notons, que le modèle peut être récursif : à son tour, l'agent élu peut re-décomposer en sous-tâche.

Bien que le CNP constitue un protocole simple basé sur le mécanismes d'appels d'offres, il n'en

demeure pas moins un processus de communication qui permet d'atteindre un accord et dans ce sens constitue une forme de négociation relativement simple de même que les protocoles de ventes aux enchères classiques. Depuis sa version initiale et compte tenu de son caractère assez général, le CNP a connu de nombreuses extensions. On citera entre autres les compléments apportés au protocole (tours successifs, confirmation et refus de proposition) par la FIPA [FIPA TC Communication 2002b]. Les figures 4.2 et 4.3 présentent les versions standardisées du CNP et du CNP étendu. Initialement, le CNP a été appliqué pour l'allocation de ressources ainsi que dans des systèmes de surveillance distribuée tels le contrôle de trafic aérien [Smith et Davis 1980]. Il a aussi été utilisé dans [Malone et Howard 1983], comme protocole d'ordonnancement distribué pour l'allocation de tâches de travail entre stations d'un même réseau local. Par la suite, d'autres travaux ont prolongé le CNP en y intégrant des contraintes globales [Conry *et al.* 1988], la gestion de l'incertitude dans les systèmes de commerce électroniques [Sandholm et Lesser 1995], ainsi qu'une implémentation du CNP, enrichie par des mécanismes de coûts marginaux [Sandholm 1993].

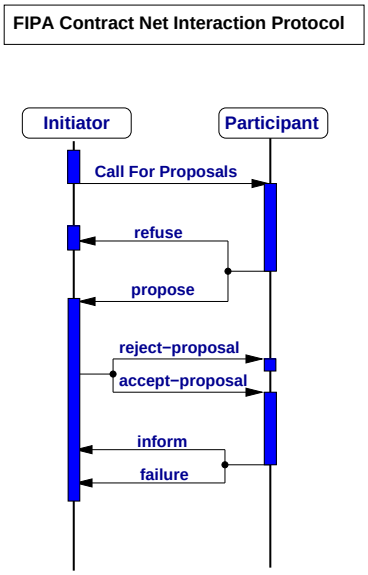


FIGURE *4.2 – Actes de langages dans le Contract Net Protocol.*

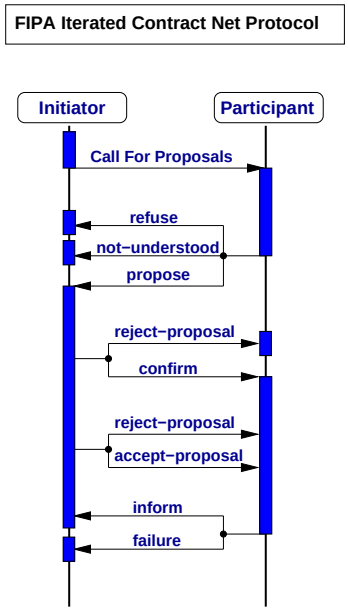


FIGURE *4.3 – Actes de langages dans le Contract Net Protocol étendu.*

Dans un tout autre domaine d'application, les enchères sont devenues un mécanisme d'achat et de vente à grande échelle avec l'explosion du commerce électronique. De nombreux protocoles d'enchères, largement inspirés des mécanismes d'enchères des sociétés humaines, ont alors été introduits mais les mécanismes d'interactions restent assez simples. Ils comprennent un initiateur qui annonce l'objet à vendre et plusieurs participants intéressés. L'initiateur veut vendre l'objet au prix le plus élevé et les participants veulent l'acheter au prix le plus bas. Dans ce cas, la négociation a pour but de trouver le prix qui convient aux parties. Il existe plusieurs protocoles d'enchères (anglaises, première offre-cachée, hollandaises, Vickrey). Pour avoir un aperçu de quelques applications de la négociation à la vente aux enchères électroniques, on pourra se référer aux systèmes eBay's Auction Web [eBays's Auction (Web Site)], FishMarket Auction House [FishMarket (Web Site)], eAuctionHouse [eAuctionHouse (Web Site)] ou AuctionBot [Wurman *et al.* 1998]. Enfin, d'autres protocoles, non spécifiques à la négociation, ont été proposés. Ce sont des protocoles d'enregistrement qui permettent de mettre en correspondance des agents fournisseurs à des agents consommateurs, ou des des protocoles d'apprentissage.

Raisonnement des agents

Le troisième aspect à prendre en compte lorsque l'on souhaite modéliser des systèmes de négociation automatisés est le raisonnement des agents. En effet, au cours de la négociation, les agents doivent prendre des décisions et choisir les actions appropriées dans le but d'accomplir leurs objectifs. Les agents doivent ainsi avoir les capacités à *analyser et apprécier une requête*, et à *construire une réponse appropriée*. Pour cela, ils doivent pouvoir s'appuyer sur des modèles de décision qui tiennent compte principalement de leurs buts et de la nature des objets négociés et leur permet de réaliser les actions adéquates. Les modèles de décision peuvent devenir sophistiqués, par exemple, en prenant en compte d'autres données obtenues par observation des actions ou des décisions réalisées par les autres agents, ou en intégrant des mécanismes d'argumentation de façon à modifier le point de vue des autres agents.

Une stratégie fine et adaptée à l'application détermine fortement l'issue de la négociation et la vitesse de convergence vers le point d'accord. Ainsi, dans la plupart des modèles, les processus de décision sont prédéfinis et fortement couplés à l'application. Ils peuvent donc être bien adaptés mais ils dépendent clairement de l'application cible. On peut distinguer les stratégies simples qui font intervenir des fonctions de type linéaires ou quadratiques (ou n'importe quel autre type de fonction cablée) [Chavez et Maes 1996] qui ne conviennent généralement qu'aux négociations sur un seul critère donné ; et les stratégies plus complexes et plus évoluées de Jennings [Faratin *et al.* 1999; Sierra *et al.* 1997; Faratin 2000] qui mettent en oeuvre des fonctions d'évaluation (fonction additive pondérée multi-critères) et de proposition (concession, donnant-donnant, rajout-suppression). Par exemple, dans les ventes aux enchères, les stratégies des agents, vendeurs et acheteurs, sont très souvent modélisées par des fonctions de décision simples prédéfinies de type linéaire ou quadratique et qui ne concernent qu'un seul critère de négociation. D'autres façons de modéliser les capacités internes de raisonnement des agents ont été introduites par Rosenschein et Zlotkin [Rosenschein et Zlotkin 1994] et se basent sur les fonctions d'utilité issues des travaux de la Théorie des jeux.

Dans le contexte de la négociation, les fonctions d'utilité représentent les gains liés à la réalisation des actions. Les utilités sont généralement représentées par des fonctions, et des matrices de décision permettent aux agents de consulter le gain lié à la réalisation d'une action donnée. Ainsi, à chaque tour, la stratégie de l'agent consiste à optimiser ses gains selon sa rationalité (minimiser le prix d'achat dans les enchères, maximiser sa consommation en ressources). Les travaux les plus explicites à ce sujet proviennent de [Faratin *et al.* 1999; Sierra *et al.* 1997; Faratin 2000]. Dans ces travaux, on distingue clairement l'évaluation des propositions de la formulation des propositions. L'évaluation des propositions se fait par le biais d'une fonction additive, qui pour chaque critère, permet d'apprécier la valeur reçue et qui pondère chaque critère en fonction de leur importance. Pour formuler des propositions, trois mécanismes sont introduits :

1. **la concession** : elle repose sur des fonctions décroissantes (polynômiales ou exponentielles) dont la vitesse de décroissance peut dépendre du temps (plus longtemps on négocie, plus on est enclin à concéder), des ressources ou par observation de la manière dont les autres font leurs concessions.
2. **le donnant-donnant** (*trade-offs*) : il consiste à formuler une proposition différente de celle reçue mais qui reste équivalente dans le sens où l'évaluation, par une fonction de similarité, des deux propositions est égale. En d'autres termes, pour X_1 une proposition reçue, $\exists X_2$ une autre proposition et S une fonction de similarité telles que $X_2 \neq X_1$ et $S(X_2) = S(X_1)$.
3. **le rajout-suppression de critères** (*issue changes*) : il se base sur la modification de la liste des critères sur lesquels portent la négociation. Par exemple, on peut retirer de cette liste les critères moins importants ou sur lesquels bute la négociation. À l'opposé on peut aussi y rajouter de nouveaux critères qui apportent des arguments supplémentaires pour relançer la négociation.

Il est aussi possible de combiner ces trois stratégies en définissant une méta-stratégie qui permet à un

moment donné de choisir un des trois mécanismes précédents. D'autres travaux [Balogh *et al.* 2000] proposent un mécanisme de concession, beaucoup plus simple et qui s'appuie sur la notion de préférences. Il se base sur une liste d'alternatives qui représente la liste des concessions faites successivement au cours de la négociation. Cette liste d'alternatives peut être définie de manière discrète via un ensemble prédéfini ou de manière continue via une fonction décroissante. Les différentes capacités de décision que nous venons de présenter constituent de *bonnes* stratégies pour faire avancer la négociation en présence d'informations limitées. Pour les fonctions de décision simples, de type linéaire ou quadratiques, l'optimum est calculable. En revanche, pour d'autres fonctions moins simples, des problèmes peuvent se poser au niveau de l'expression de la fonction de décision mais aussi pour les calculs d'optimisations. En ce qui concerne les trois mécanismes évoqués, les stratégies de donnant-donnant ou de rajout-suppression de critères reposent sur des mécanismes de recherche parmi un ensemble calculé, elles sont donc sujettes à des problèmes de complexité calculatoire et allongent d'autant la stratégie de la négociation. La stratégie de rajout-suppression de critères, quant à elle, présente de bonnes qualités pour faire avancer une négociation qui ne converge pas. En revanche, ils présentent des problèmes au niveau de la sémantique de la négociation. En effet, pour avoir une stratégie réellement efficace les agents doivent savoir quelles manipulations faire sur les critères. Pour le donnant-donnant, il s'agit de déterminer les critères qui sont à mettre en balance, alors que pour le rajout-suppression, il faut avoir une idée des critères qui sont à ajouter ou à supprimer de la négociation. À ce titre, et indépendamment des travaux de Jennings, un modèle de négociation à deux niveaux a été défini pour prendre en compte *i)* la manière dont les agents doivent négocier (e.g les actions à entreprendre) et *ii)* les manipulations que les agents doivent faire sur les critères à négocier pour orienter leur stratégie de négociation [Chevrier 1992]. Ce modèle de négociation à deux niveaux donne aux agents des capacités à gérer explicitement leurs processus de négociation et les rend ainsi plus autonomes.

Pour finir, nous évoquons simplement les problèmes qui peuvent apparaître au moment de l'implémentation du modèle de négociation dans la mesure où ces stratégies reposent essentiellement sur la définition de fonctions. En particulier, il s'agit de préciser les personnes responsables de la définition de ces fonctions, la manière de les définir simplement et aussi la manière de les implémenter.

4.2.3 Systèmes de négociation

Parmi les systèmes de négociation existants, de nombreux travaux relèvent des mécanismes de marché. Ce domaine devient ainsi de plus en plus attractif pour la communauté des systèmes multi-agents, non seulement en raison de la disponibilité de modèles formels sous-jacents mais aussi pour son applicabilité directe aux transactions commerciales basées sur Internet dont l'importance actuelle est vouée à croître.

Places de marché électroniques

Kasbah Kasbah [Chavez et Maes 1996] développé au MIT Lab par Maes est un système dans lequel des agents négocient la vente et l'achat de produits sur Internet. Dans ce système, des agents de type vendeur et acheteur sont crés pour représenter les acheteurs et les vendeurs, et diverses informations sont associées à leur création (produits concernés, prix désirés, date d'expiration, etc.). La négociation est de cardinalité 1-N et fait intervenir un vendeur et plusieurs acheteurs après que ces derniers aient été mis en correspondance par un système similaire aux annonces. Elle se déroule selon plusieurs tours de sorte que le vendeur puisse comparer les propositions d'achat successives des acheteurs. Trois stratégies de négociation différentes prédéfinies *anxieuse*, *enthousiaste* et *frugale* peuvent être utilisés pour paramétrer les agents acheteurs. Elles correspondent respectivement à des fonctions linéaires, quadratiques et exponentielles leur permettant de guider leurs offres successives. Le projet Tete-@-Tete [Guttman et Maes 1998]

a été initié pour étendre les capacités de Kasbah en intégrant la négociation multi-critères, l'inclusion de garanties et de délais de livraison ainsi que d'autres aspects liés aux qualités des transactions.

AuctionBot AuctionBot [Wurman *et al.* 1998] est une autre infrastructure développée à l'Université de Michigan, qui propose un serveur d'enchères. Elle permet aux agents de participer à divers types d'enchères (anglaise, hollandaise,Vickrey,...). Les agents débutent les enchères dans le système jusqu'à ce que l'enchère soit fermée puis rendent compte des résultats à leurs mandants. Du point de vue de l'implémentation de la négociation, ce système est dédié aux enchères et les allocations des enchères sont réalisées par le serveur en fonction d'un ensemble de règles bien défini. Il n'y a donc pas réellement de communication pour parvenir à un accord.

FishMarket Fishmarket [Rodríguez-Aguilar *et al.* 1998] est un autre système d'enchères électroniques conçue dans le milieu académique. Les enchères sont inspirées des marchés de vente de poissons espagnols dans lequel les propositions décroissent (enchères de type hollandaise). La particularité de ce système est que des agents mandatés peuvent être amenés à négocier avec des utilisateurs humains directement au travers d'applets Java.

Magnet Le système Magnet (*Multi AGent Negotiation Testbed* en anglais) [Collins *et al.* 1998a,b] a été conçu à l'Université du Minnesota pour pallier l'absence d'architectures permettant de prendre en compte divers protocoles de négociation. Pour cela, le système Magnet propose de prendre en compte une grande variété de négociation de l'achat/vente de biens à la négociation de contrats complexes entre agents. En particulier, le but est de proposer une architecture généralisée qui fournit un support pour les transactions commerciales de type B2B (*business-to-business*). Les agents négocient et surveillent l'exécution des contrats commerciaux qui les lient aux différents fournisseurs. Chaque agent est une entité indépendante possédant sa propre structure, ses buts et ses ressources. Toutefois, les ressources dont disposent les agents ne sont, généralement, pas suffisantes pour l'accomplissement de leurs buts. Ainsi, la négociation porte à la fois sur la réalisation de services et sur les ressources nécessaires à la réalisation de leurs buts. Dans la négociation, les rôles de clients et de fournisseurs sont distingués, la négociation porte essentiellement sur le prix des objets et incluent d'autres éléments (pénalités, dates limite de réception des offres, de notification des réponses,...) et le protocole de négociation entre clients et fournisseurs fonctionne selon les mécanismes d'appel d'offres et de façon très proche du *Contract-Net Protocol*. Ainsi, les contributions du système reposent davantage sur la définition générale d'une architecture de marché que sur le moteur de négociation lui-même. Les éléments importants du marché incluent une ontologie spécifique, des définitions de types de protocoles de négociation supportés, un registre de clients ainsi que des mécanismes pour gérer les transactions (intermédiaire explicite qui contrôle les engagements, sessions, protection contre la fraude,...).

SilkRoad Le projet SilkRoad [Ströbel 2001], développé au Laboratoire d'IBM Zürich, se concentre aussi sur les négociations commerciales. Ce système a pour but de faciliter la négociation dans un contexte *e-business* en proposant une méthodologie spécifique de conception et une architecture générique. Pour réaliser cela, le système s'appuie sur un système permettant de découvrir les articles disponibles et de mettre en correspondance, par appariement, différents articles par la donnée des contraintes sur leurs attributs [Michael et Markus 2001]. Par la suite, le système de négociation agit comme un intermédiaire entre toutes les parties de la négociation (agents logiciels ou humains). L'ensemble des règles de communication et des stratégies de négociation est complètement intégré et définit dans cet intermédiaire. Une classification extensive des systèmes de négociation adaptés au commerce électronique est proposée dans [Lomuscio *et al.* 2001].

Systèmes génériques

Plate-formes de négociation générique La plate-forme générique de négociation (*Generic Negotiation Platform* en anglais) [Benyoucef *et al.* 2000], proposée à l'Université de Montréal, se place aussi dans le cadre des enchères électroniques. La négociation entre les agents s'effectue au travers d'un service de négociation qui permet d'apparier les demandes et les offres des acheteurs/vendeurs, et de fixer les prix. Dans le GNP, des acteurs et de rôles son définis (lecteur, annonceur, négociateur, administrateur), les règles de la négociation sont décrits séparément dans des documents et la négociation se déroule par envoi de documents (XML, HTML).

D'autres travaux de Bartolini et al. [Bartolini *et al.* 2002] ont aussi conduit à l'élaboration d'une plate-forme générique de négociation automatisée. L'objectif de ces travaux est d'offrir une alternative au développement usuel de protocoles dédiés à certaines applications cibles en proposant un protocole d'interaction simple, potentiellement utilisables dans différentes circonstances. Pour cela, il introduit avantageusement différentes règles de négociation qui peuvent être utilisées pour paramétrer la plate-forme et ainsi implémenter des mécanismes de négociation différents. Ainsi, au cours de la négociation, les participants échangent des propositions qui doivent être valides en spécifiant certains paramètres de la négociation (type de produit, prix, date de livraison,...) et respecter les règles de la négociation (participants autorisés à proposer, validité des propositions par rapport à l'état courant de la négociation,....). Ces propositions sont converties en accord dès qu'un accord est formé et la négociation se termine selon les règles de terminaison spécifiées. Deux rôles sont aussi introduits dans cette plate-forme, les participants qui veulent aboutir à un accord et postent leurs différentes propositions, et l'hôte de la négociation, qui peut être éventuellement un participant, chargé de faire gérer la négociation : création et surveillance des règles de la négociation, exécution de la négociation, résolution de l'accord et finalisation de la négociation. Une implémentation de cette plate-forme a été réalisée avec le système *Jade* et des envois de messages au format FIPA-ACL.

Zeus Zeus [Nwana *et al.* 1999] est une boite à outils pour la programmation d'agents. Il ne se concentre pas sur un type d'application en particulier mais propose davantage un ensemble d'outils et de bibliothèques Java, permettant de développer des agents dans une large variété de problèmes. L'architecture de Zeus se décompose en trois couches : la couche de définition permettant de définir le fonctionnement interne des agents (compétences, rationalité, ressources, croyances,...), celle d'organisation qui considère les relations entre les agents (connaissance d'autres agents, capacités des autres agents connues), et la couche de coordination qui permet de considérer les interactions des agents, en particulier en terme de coordination et de négociation.

Genca Le modèle *GeNCA* propose une API Java permettant l'implémentation de systèmes de négociation génériques [Mathieu et Verrons 2002; Verrons 2004]. Ainsi, à partir d'une étude des systèmes de négociation existants, le modèle générique de négociation réifie certains éléments de base (ressources, personnes négociantes, actes de langages minimaux pour une négociation) et structure le modèle générique selon les trois niveaux de *communication*, de *négociation* et de *stratégie*. La couche de communication définit entre autres le support des communications et les types et formats des messages, la couche de négociation introduit entre autres les contrats et le négociateur, et enfin la couche de stratégie pour mettre en oeuvre le raisonnement des négociants. Dans *GeNCA*, le niveau communication permet d'abstraire le processus de communication et n'est pas imposé. En effet, elle peut, par exemple, s'appuyer sur une plate-forme à agents et ainsi utiliser le support de communication de celle-ci ou alors, être réalisée par communication de processus ou échanges d'email. Les communications peuvent se faire soit entre les agents, soit entre les agents et le serveur de noms dans le but de regrouper des agents intéressés par une négociation. Cela induit ainsi deux types de messages destinés à représenter les échanges avec le

serveur ou entre les agents. Enfin, les messages de négociation définissent un émetteur, une primitive, des éventuels paramètres ainsi que l'identifiant et l'état de la négociation.

Le niveau de négociation distingue les objets permettant de *(i)* décrire la négociation (ressources négociés, contrats engagés), *(ii)* gérer la dynamique de la négociation (négociateur, micro-agents) et *(iii)* mémoriser les négociations passées pour, par exemple, établir des statistiques de succès ou de rétractation par couple (participant,ressource). Dans ce modèle, une *ressource* décrit ce que chaque agent tente d'obtenir et les *contrats* sont créés à l'initialisation des négociations et encapsulent les ressources à négocier, l'identifiant de l'initiateur, un délai de réponse et une réponse par défaut. Enfin, les *propriétés* du contrat décrivent certains aspects du déroulement de la négociation (accords minimaux pour conclure un contrat, participants impliqués, nb de re-négociations autorisées,...). Au niveau du déroulement de la négociation, un *négociateur* gère toutes les négociations d'un agent donné, il connaît tous les contrats initiés, leurs propriétés et leurs résultats. Chaque agent *négociateur* est constitué de *micro-agents*, chacun étant purement réactif et responsable de la négociation d'un contrat donné. Ces micro-agents représentent ainsi soit l'initiateur, soit un participant et ils interagissent entre eux avec une stratégie qui leur a été attribuée. Enfin, le niveau stratégie s'appuie sur deux interfaces définissant l'ensemble des fonctions qui permettent la prise de décision. Ainsi, l'initiateur peut réagir à la réception d'un message d'acceptation, à la réception d'une (ou de toutes les) modification(s) et peut élaborer une décision à partir de toutes les réponses reçues. Les participants, quant à eux, peuvent réagir à la réception d'un contrat et proposent à l'initiateur une modification du contrat. Le modèle *GeNCA* intègre une implémentation, par défaut, pour chacune de ces interfaces de stratégie, néanmoins pour des applications spécifiques, le développeur peut implémenter de nouvelles stratégies. Des exemples d'application du modèle générique à des cas usuels de négociation ont été proposées dans [Mathieu et Verrons 2004].

4.3 Synthèse

Dans ce chapitre, nous avons tout d'abord présenté rapidement le contexte dans lequel un processus de négociation peut s'enclencher. La négociation peut être vue sous sa forme la plus basique et succinte comme « *un processus de communication entre un groupe d'entités dont le but est d'atteindre un accord mutuel à propos de certains éléments* » [Bussmann et Muller 1992]. Ces entités sont plongés dans un environnement partagé. Ils ont des buts à atteindre, échangent des informations, et se coordonnent pour réaliser des objectifs. Compte tenu de sa généralité, diverses formes de négociation peuvent toutefois être élaborées et vont se distinguer par les rôles tenus par les participants, leurs communications, leurs capacités de raisonnement au cours de la négociation, et le processus général de déroulement. Par ailleurs, d'autres approches alternatives à la négociation existent. Les plus proches de la négociation sont les structures organisationnelles, la contractualisation et la planification qui se focalisent sur la coordination et introduisent sous une forme générale, des parties qui interagissent pour résoudre des problèmes, alors que le vote, la théorie des jeux, et la satisfaction de contraintes sont plus éloignés et davantage orientés vers la détermination de solutions aux problèmes, sous des hypothèses bien précises.

Les principales applications de la négociation sont réalisées dans les systèmes multi-agents (SMA). Les SMA ont été créés pour résoudre des problèmes informatiques par des collaborations intelligentes, inspirées de certains raisonnements humains, entre des entités actives et distribuées. L'étude montre que c'est bien le domaine ou les capacités de négociation sont les plus diverses et les plus abouties. Etant donnée l'autonomie des agents, la négociation est un processus essentiel dans les SMA. Les différentes composantes de la négociation définies dans le chapitre précédent y sont donc particulièrement variées et détaillées (places de marchés, négociations adaptées du *Contract-Net Protocol*). Ce dernier protocole, déjà adapté à diverses applications, semble le plus générique et donc, il constitue celui qui pourrait donner les premières bases pour un processus général de négociation de contrats. Parmi les raisonnements

établis par les agents, seuls les plus simples semblent adaptables à notre problématique, et en particulier, les notions de concession et de donnant-donnant peuvent trouver leurs équivalents dans l'expression de propriétés sur des entités logicielles.

5
Bilan de l'état de l'art

Ce chapitre dresse un bilan de l'état de l'art. Il effectue une analyse et tire les principales conclusions des travaux étudiés dans les chapitres précédents. Puis, il présente le cahier des charges et les qualités attendues des processus et du modèle de négociation.

5.1 Conclusions sur les travaux étudiés

Les chapitres précédents ont présenté les principaux travaux autour des contrats dans les composants, des aspects extrafonctionnels et de la négociation dynamique. Le premier chapitre a brièvement abordé les contrats dans les composants et les infrastructures qui les supportent. Il apparaît que la plupart des propositions qui traitent les contrats dans les composants se focalisent surtout sur les aspects fonctionnels (protocole, comportement, substituabilité...) dans le contexte des interactions client-serveur des composants. Par ailleurs, le modèle de contrats de *ConFract* présente des caractéristiques très intéressantes compte tenu, notamment, des possibilités de spécifier diverses propriétés sur différents éléments architecturaux des systèmes (interfaces, composants) et à des portées différentes (connexion entre interfaces, portée interne ou externe des composants), et de vérifier dynamiquement les contrats construits en fonction de leur environnement d'évaluation. La comparaison des caractéristiques du modèle *ConFract* avec les autres travaux présents dans la littérature permet ainsi de valider notre hypothèse de travail d'utiliser cette plate-forme.

Le second chapitre a proposé une vision complète des travaux sur les aspects extrafonctionnels. De par leur diversité, ces derniers ne sont pas abordés dans une approche complète et la plupart des travaux se concentrent sur certaines dimensions des aspects extrafonctionnels. Il n'y a pas d'approche systématique pour décrire et délimiter le champ des aspects extrafonctionnels, et cela se fait de façon empirique, au cas par cas en capitalisant les expériences acquises dans les études d'applications ou de classes d'application bien précises. Pour spécifier les aspects et propriétés extrafonctionnels des systèmes, de nombreux langages de spécifications ont été proposés et les plus généraux introduisent des concepts clés pour organiser les spécifications et leur mise en correspondance sur les interfaces des objets ou composants. Toutefois, les éléments spécifiés dans ces langages restent à des niveaux très élevés. Les spécifications concernent des définitions de niveaux de qualité acceptables exprimés par des intervalles de valeurs valides, et elles ne permettent souvent pas de référencer des éléments applicatifs de la plate-forme, à l'exécution. Cela est

particulièrement préjudiciable dans le cadre de notre travail car l'adaptation dynamique doit surveiller activement les systèmes et leur environnement lors de leur exécution. De plus, lorsqu'ils sont évoqués, les mécanismes de gestion des spécifications sont simples et s'appuient sur des structures de données ou des objets spécifiques générés de façon *adhoc*. Les relations de ces derniers avec les spécifications initiales ne sont donc pas claires, et cela a pour conséquence directe de limiter les possibilités de gestion fine des spécifications à l'exécution, ainsi que la mise en œuvre de mécanismes d'adaptation des systèmes qui s'appuient fortement sur des notions contractuelles. De ce fait, dans les meilleurs cas, les vérifications effectuées sont simples et consistent à faire statiquement des tests de compatibilité de spécifications exprimant des niveaux de qualité sur des domaines de valeurs. De plus, les capacités d'adaptation et de négociation reposent sur des modifications de profils de QdS définis, et reviennent à remplacer les profils non satisfaits par de nouveaux profils qui expriment des contraintes de qualité moins fortes.

Sous une forme plus opérationnelle, d'autres travaux proposent des plates-formes de contractualisation de ressources. Elles définissent des contrats sur des ressources ainsi que des infrastructures qui permettent de gérer les contrats construits. Toutefois, les fonctionnalités réalisées reviennent à effectuer de la réservation des niveaux de ressources spécifiés et à surveiller leurs utilisations lors de l'exécution. L'inconvénient majeur de ces propositions réside dans la granularité des mécanismes. Bien souvent, les ressources sont exprimées par des profils d'utilisation qui imposent des seuils d'utilisation fixes. De plus, les spécifications et les surveillances portent sur l'application globale et ne considèrent pas des éléments plus fins tels que les éléments (composants) qui la constituent. Les violations de contrats sont donc souvent traités par des signalisations, et les quelques propositions d'adaptations reviennent à modifier les profils d'utilisation. Nous estimons que c'est un inconvénient majeur car cela limite les possibilités de raisonner de façon plus fine pour répartir l'auto-adaptation au niveau des sous-éléments des applications. D'autres travaux se concentrent sur la mise en œuvre de techniques de surveillance et d'auto-adaptation dans les systèmes à composants. Les logiques d'adaptations sont séparées de la spécification des systèmes eux-mêmes, définies par des règles ECA ou des automates, et intégrées par diverses techniques (tissage d'aspects, intercession au méta-niveau des composants...). Ces logiques d'adaptation appartiennent à la famille des *politiques basées actions* et spécifient clairement l'action à effectuer lorsque le système ou l'environnement est dans un état donné. Ainsi, les actions reviennent souvent à modifier l'implémentation courante des composants, à faire des modifications structurelles ou à modifier des attributs de composants. De plus, dans les systèmes réflexifs, les adaptations sont effectuées de façon synchrones en étant couplées à l'exécution des opérations du niveau de base des composants. Enfin, il est à noter aussi que certaines techniques offrent maintenant la possibilité de mettre à jour et modifier les logiques d'adaptations aussi à l'exécution des systèmes. Compte tenu de la complexité du dénombrement de tous les états possibles, il n'y a jamais de stratégies *complètes* (ie. spécifiant toutes les actions sur tous les états) et les logiques définies se limitent à une couverture partielle des cas évalués comme pertinents, par les développeurs ou administrateurs. De plus, les techniques d'adaptations mises en œuvre n'intègrent pas de règles d'intégrité ou de cohérence, si ce n'est certaines vérifications structurelles (interfaces connectées, dénombrement de sous-composants, contraintes d'architecture...).

Enfin, les travaux les plus complets qui s'appuient à la fois sur des technologies de contrats et d'adaptations sont les plus proches de notre problématique. Ils définissent souvent leur propre infrastructure pour gérer les contrats, et les objets définis qui servent de support aux adaptations restent très spécifiques aux langages de spécification et à la plate-forme d'exécution. De plus, l'adaptation est souvent réalisée dans le contexte d'interactions client-serveur distribuées, et les actions d'adaptations consistent à modifier des contrats (profils de QdS) et changer les implémentations de composants. Les techniques d'adaptations sont aussi souvent organisées autour d'une entité centralisée qui a une vue globale sur les contrats et les différents éléments qui gère les contrats, et ils ne définissent pas de support pour réaliser les adaptations qui utiliserait des techniques distribuées. En ce qui concerne, les mécanismes de négociation, ceux-ci restent assez grossiers et consistent plus en des processus de recherche ou des tests sur

les différentes configurations ou implémentations possibles qui satisfont les besoins de QdS demandés. Les adaptations restent classiques et souvent exprimées par des règles ECA. Toutefois, les cohérences des adaptations sont à la charge des développeurs.

Enfin, vis-à-vis de notre objectif de mettre en œuvre des mécanismes de négociation entre composants contractualisés, le troisième chapitre à présenté une vision d'ensemble des principes généraux de la négociation dynamique, puis à identifié les principales composantes de négociation dans les systèmes multi-agents (SMA). Les formes les plus décentralisées des négociations identifient ainsi des parties négociantes qui vont interagir sur des objets de négociation sur la base d'un protocole. Dès lors, le modèle de négociation de contrats proposé devra expliciter au mieux toutes les différentes composantes de la négociation, en s'inspirant des modèles existants notamment dans les systèmes multi-agents, tout en l'adaptant aux mieux aux spécificités des contrats logiciels. Vis-à-vis de notre problématique, il apparaît déjà que comme les composants sont porteurs de contrats, ils devraient être assimilables aux participants d'un négociation si un contrat est violé. Dans les SMA, les agents négocient le plus souvent la répartition de tâches à accomplir avec des capacités de raisonnement inspirées des êtres vivants. Dans notre cas, la négociation va plutôt porter sur les contrats lorsqu'ils sont violés. En revanche, les aspects relatifs aux capacités de communication et de raisonnement sont très évolués dans les SMA et devront être davantage explicités dans nos mécanismes de négociation.

5.2 Cahier des charges

À partir de notre problématique déterminée dans l'introduction de ce document, cette section établit le cahier des charges du modèle de négociation et discute les différentes décisions de conception vis-à-vis des besoins exprimés. Ces décisions sont motivées par le fait que le modèle à concevoir doit s'adapter à la fois à la nature des composants, des contrats et des contraintes à négocier.

Négociation des aspects extrafonctionnels. De façon générale, les contraintes exprimées dans les contrats peuvent porter sur des aspects fonctionnels et extrafonctionnels. Dans notre étude, nous considérons que les aspects fonctionnels ne peuvent être que très peu modifiés, et qu'ils doivent plutôt être gérés avant l'exécution du système ou lors des phases de tests et d'intégration. De plus, bien que la détection des erreurs fonctionnelles lors de l'exécution est importante, il n'y a que très peu de marge de manœuvre pour les gérer à l'exécution. Ainsi, la négociation des aspects fonctionnels n'est pas un objectif de notre modèle. En revanche, les mécanismes de négociation vont se concentrer sur les contrats qui spécifient des aspects extrafonctionnels, et devront permettre de trouver un nouvel état d'équilibre qui satisfasse les diverses contraintes extrafonctionnelles.

Intégration au cycle de vie des composants. Compte tenu des différentes phases du cycle de vie des composants, et des différentes propriétés que peuvent spécifier les contrats, la négociation doit s'intégrer correctement dans le cycle de vie des composants et de leurs contrats. En particulier, la négociation aura surtout lieu à l'exécution des systèmes lorsque les contraintes spécifiées ne sont pas satisfaites, mais, en fonction de la nature des propriétés négociées, elle pourra s'effectuer à d'autres moments si besoin (phase de configuration, phase de déploiement...). Par ailleurs, bien que de nombreux travaux utilisent la négociation pour établir un contrat initial, nos mécanismes de négociation vont davantage travailler sur des contrats existants mais qui sont violés par les variations de fonctionnement du système ou de l'environnement. Ainsi, la négociation se concentre plus sur le rétablissement de la validité des contrats que sur leur construction.

Identification des composantes de base. Compte tenu de la richesse de la notion de négociation, le modèle devra clairement identifier les éléments de base qui sont nécessaires et suffisants pour démarrer un processus de négociation. Il pourra, pour cela, s'appuyer sur les composantes de base

(objets, parties et protocole de négociation) usuellement définies dans les mécanismes de négociation dans les systèmes multi-agents, mais en les adaptant aux spécificités des contrats, des composants logiciels, et de la problématique de l'auto-adaptation. Conformément à notre démarche, les parties négociantes de nos processus seront les composants, mais comme ils ne sont pas *a priori* tous concernés par la négociation d'un contrat donné, le modèle devra permettre une identification plus précise des parties négociantes effectives pour chaque contexte de négociation. De plus, comme la négociation vise à exploiter les capacités internes des composants, le modèle devra clairement définir *i)* les informations et capacités attribuées aux composants ainsi que *ii)* les mécanismes employés pour qu'ils puissent interagir au cours des processus de négociation.

Exploitation des contrats. Pour exploiter les capacités des entités contractualisées à organiser l'auto-adaptation, la négociation doit en particulier s'appuyer sur des informations fines des composants vis-à-vis des clauses des contrats négociées, telles que les rôles ou responsabilités des composants. Cela permettra non seulement de faire cibler les interactions entre des entités précises mais aussi d'offrir un premier niveau qui permet d'orienter la négociation, tout en évitant de consulter d'autres entités non impactées. De plus, lorsque les contrats définissent des éléments atomiques telles que les clauses, la granularité des négociations devra aussi se faire, au grain le plus fin, au niveau de ces éléments atomiques. Le modèle de négociation devra donc définir des mécanismes de négociation qui s'adressent à ces éléments atomiques des contrats, et expliciter au mieux les portées spatiales et temporelles de ces négociations.

Pilotage par des politiques de négociation. Comme le processus de négociation est potentiellement très ouvert et qu'il va faire intervenir des composants comme parties dotées de leur propre capacités, il est nécessaire d'intégrer au processus de négociation, des *politiques de négociation*, qui vont permettre d'orienter et de maintenir un minimum de contrôle dans les déroulements des négociations. En fournissant diverses orientations possibles, de telles politiques vont par ailleurs aussi enrichir les processus de négociation en présentant différentes formes de déroulement possibles.

5.3 Qualités attendues

Nous distinguons les qualités attendues par les processus de négociation de celles attendues par le modèle lui-même.

5.3.1 Qualités du processus

Nous déterminons ici les qualités attendues pour le processus de négociation lui-même. Nous nous intéressons au déroulement de la négociation entre les parties, plutôt qu'aux éléments ou acteurs de cette négociation :

Simplicité. La première qualité du processus est sa simplicité. Cela est nécessaire pour qu'il puisse être configuré et utilisé par des développeurs et des administrateurs de système qui seront amenés à contractualiser, de façon négociables, un nombre important de propriétés extrafonctionnelles, voire fonctionnelles, sur un ensemble de composants.

Raisonnement privé. Comme les composants sont équipés de mécanismes pour interagir et négocier, leur manière de négocier doit rester privée. Seules devront être éventuellement visibles leurs diverses propositions selon le processus de la négociation. Cette propriété est souvent recherchée dans les systèmes multi-agents et s'applique à tout système distribué où des intérêts potentiellement divergents cohabitent.

Ouverture vs. contrôle. Comme la négociation de contrats conduit à adapter dynamiquement les composants. Il est envisageable qu'une négociation entraîne des reconfigurations plus ou moins fortes de composants. Il faut alors caractériser quelles sont les capacités d'adaptation — plus elles sont larges, plus le modèle sera puissant — et quel degré de contrôle attend-t-on sur ces adaptations — plus les adaptations sont puissantes, moins elles sont potentiellement contrôlables dans leur cohérence. Les qualités d'ouverture et de contrôle seront donc opposées et il faudra étudier le processus fourni selon les deux points de vue. Par exemple, la *convergence* du processus peut être vue comme une caractéristique de contrôle, mais elle n'est même pas forcément assurée dans d'autres processus d'adaptation dynamique.

Stabilité et reproductibilité. La stabilité du processus se définit par la capacité à prévoir ses performances. Dans le cas du processus de négociation, l'environnement des composants peut varier de façon à ce que le processus ne soit même pas reproductible, puisque les composants peuvent modifier leur décision dans le processus en fonction de paramètres invisibles au niveau de la réalisation du processus. Les conditions de reproductibilité du processus devront être néanmoins déterminées au plus fin. A priori, déterminer la stabilité du processus demanderait une étude statistique très poussée.

Efficacité et optimalité. Bien entendu, il sera intéressant d'évaluer l'efficacité théorique et pratique du processus de négociation. Le caractère optimal de tout ou partie du processus pourra aussi être déterminée. De manière pragmatique, la minimisation des communications dans le processus devrait permettre de fournir une bonne efficacité dans les communications, mais ceci devra être validée par des évaluations qualitatives plus approfondies.

5.3.2 Qualités du modèle

Le modèle proposé devra exposer les qualités suivantes :

Extensibilité. Chaque élément du modèle devra être potentiellement remplaçable pour adapter, améliorer le modèle et le processus proposés par la suite.

Applicabilité. Le modèle devra être potentiellement applicable à d'autres architectures que *Fractal* et d'autres approches contractuelles que celles proposées dans *ConFract*. Il sera donc nécessaire d'étudier en quoi et comment les différents éléments du modèle sont applicables à d'autres environnements.

Adéquation à Fractal et ConFract. Comme la plate-forme de contrats choisie par hypothèse est *ConFract*, le modèle proposé devra aussi s'intégrer au mieux dans cette plate-forme et aussi exploiter au mieux ses caractéristiques. Il s'agira notamment de respecter les mécanismes mis en œuvre par *ConFract* pour la gestion des contrats (gestionnaire de contrats, mise à jour dynamique des contrats, etc.) mais aussi de respecter les principes du modèle *Fractal* utilisé dans *ConFract* (hiérarchie des composants, connexions, niveaux de visibilité, etc.).

DEUXIÈME PARTIE

CONTRIBUTIONS

Buts et portée de la partie

L'objectif de cette seconde partie est de présenter les contributions réalisées dans ce travail de thèse. Les propositions sont regroupées selon quatre chapitres. Le premier chapitre présente le modèle de négociation de contrats développé. Le deuxième chapitre décrit une contribution annexe sur l'intégration des propriétés extrafonctionnelles et un support raisonnement compositionnel, utilisée par le modèle de négociation. Le troisième chapitre présente la mise en œuvre du modèle dans les plates-formes de composants *Fractal*, et de contrats *ConFract*. Enfin, le quatrième chapitre présente une évaluation qualitative, effectue un retour sur des aspects méthodologiques et décrit les premières évaluations expérimentales des mécanismes de négociation.

6 Modèle de négociation

Dans ce chapitre, nous présentons notre modèle de négociation de contrats pour composants logiciels hiérarchiques. Dans une première partie nous présentons les principes du modèle de négociation, puis nous décrivons successivement ses éléments de base : les objets de la négociation, les parties négociantes, le protocole et les politiques de négociation. Tout le long de ce chapitre, les éléments importants du modèle sont aussi illustrés sur des exemples de contrats extraits d'un cas d'étude détaillé dans l'annexe A de ce document. Enfin, le chapitre se termine par un retour sur les choix de conception.

6.1 Principes

L'objectif du modèle de négociation consiste à réaliser l'auto-adaptation des systèmes construits à base de composants contractualisés. Comme notre approche de l'auto-adaptation se base sur des composants contractualisés, le modèle de négociation doit s'appuyer sur une plate-forme à composants contractualisés. Ainsi, nous présentons tout d'abord les principes généraux de la plate-forme *ConFract* sur lesquels s'appuie le modèle, avant de présenter ceux du modèle de négociation lui-même.

6.1.1 Principes des plates-formes sous-jacentes

Comme nous l'avons rapidement présenté dans l'état de l'art (cf. section 2.2), le modèle de contrats *ConFract* applique l'approche contractuelle a des composants logiciels hiérarchiques tels que ceux défini dans le modèle *Fractal*. Comme ce dernier définit des hypothèses très générales sur les composants et englobe les caractéristiques des autres modèles à composants existants, les principes de *ConFract* peuvent s'appliquer à d'autres modèles de composants similaires. La suite de cette section présente les principes généraux de *ConFract* exploités par le modèle de négociation. Les principes strictement nécessaires pour la négociation sont présentés ici et illustrés sur des exemples. Le lecteur intéressé pourra se référer à l'annexe B pour une présentation plus complète de *ConFract*.

Aperçu du modèle Fractal Le modèle *Fractal* [Bruneton *et al.* 2004, 2006] définit un modèle de composants général et extensible dédié à la construction et à la (re-)configuration dynamique de systèmes à granularités diverses. Dans ce modèle, les composants peuvent être de type *primitif* (lorsqu'ils encapsulent des objets simples) ou *composite* (lorsqu'ils contiennent d'autres composants). Un composant

peut aussi être *partagé* s'il est contenu dans deux composites distincts. Ces composants représentent les entités d'exécution du système et peuvent communiquer lorsque des interfaces *requises* et *fournies* sont connectées. Ces interfaces constituent alors les seuls points d'accès aux composants et correspondent à l'ensemble des services requis et fournis des composants. Le modèle propose aussi des contrôleurs pour gérer divers aspects techniques des composants (cycle de vie, assemblage, exécution...). Ainsi, des contrôleurs basiques sont déjà définis pour gérer le cycle de vie (*LifecycleController*), les connexions (*BindingController*) et le contenu (*ContentController*) des composants, et comme la plate-forme est ouverte, d'autres fonctionnalités de contrôle peuvent être ajoutées.

Les figures suivantes illustrent ces principaux éléments du modèle *Fractal* et introduisent des exemples qui seront utilisés dans la suite de ce chapitre pour illustrer les différents éléments du modèle de négociation. Ces exemples sont extraits d'une application plus générale qui est utilisée comme cas d'étude et est complètement décrite dans l'annexe A. D'un point de vue fonctionnel, cette application consiste en une application distribuée de communication instantanée dont la fonctionnalité principale consiste à former automatiquement des groupes d'utilisateurs et à leur faire partager automatiquement diverses applications en fonction de leurs centres d'intérêts (vidéo, messagerie, etc.). La figure 6.1 décrit un composant composite `InstantCommunication` qui contient trois sous-composants : `InstantGroup` possède trois interfaces fournies pour réaliser la gestion des utilisateurs et des groupes connectés dans le système, `InstantApp` qui encapsule les divers services partagés entre les utilisateurs, et `InstantFacade` qui interface les deux sous-composants précédents. En particulier, le composant `InstantGroup` possède un attribut `maxUsers` qui définit le nombre maximal d'utilisateurs concurrents dans le système.

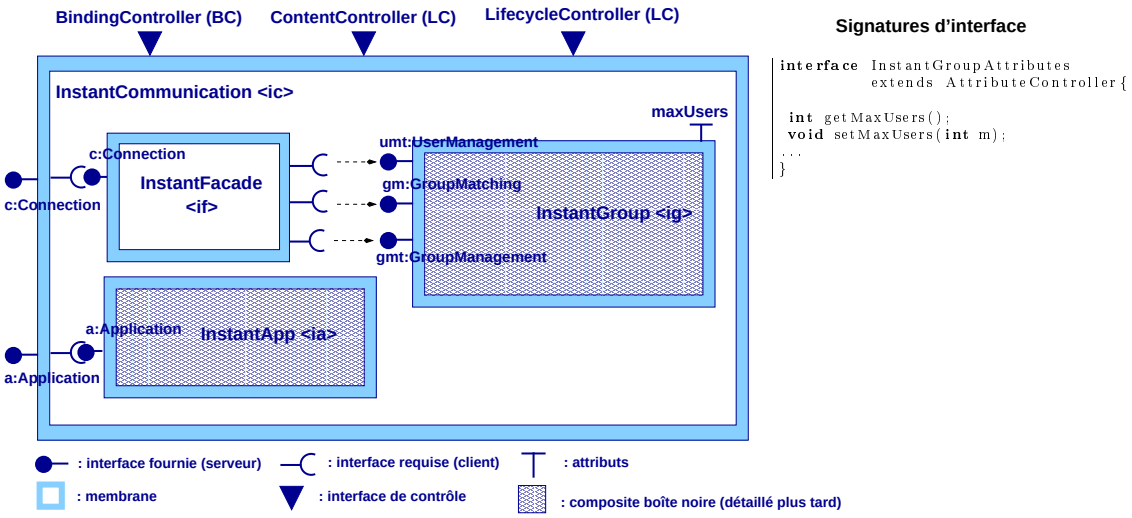


FIGURE *6.1 – Architecture en Fractal du composant InstantCommunication*

La figure A.11 détaille le contenu du composite `InstantApp`. Celui-ci est formé de trois sous-composants : `ServiceFacade` qui représente une facade vers les autres composants, `VideoService` qui établit et gère le service lié à la diffusion vidéo, et `ServiceConfigurator` qui fournit divers services pour configurer et optimiser le service précédent. En particulier, `VideoService` offre, par l'interface fournie `v :Video`, les méthodes `loadMedia` et `start`, qui permettent respectivement de charger et de débuter la diffusion vidéo, et `ServiceConfigurator` fournit une interface `Configuration` qui propose la méthode `expectedCPUUsage` permettant d'estimer l'utilisation

du CPU induite par la diffusion vidéo en tenant compte de certains paramètres comme le frameRate (nombre d'image diffusées par seconde) et le nombre de clients du service.

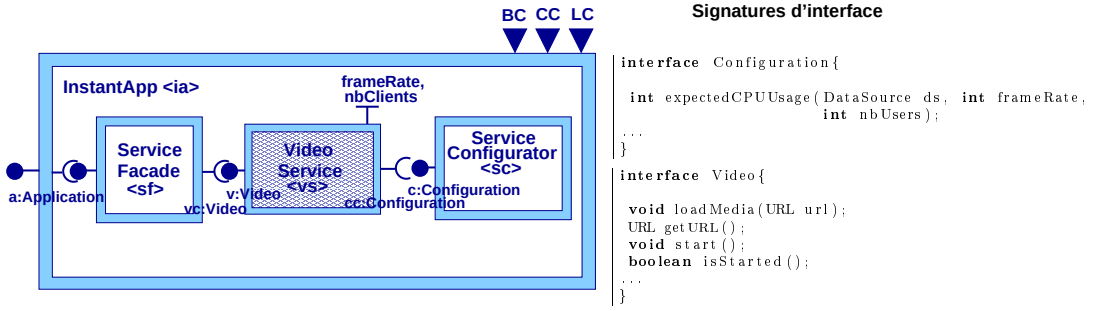


FIGURE 6.2 – *Architecture interne en Fractal du composant InstantApp*

Principaux éléments du modèle ConFract Comme présenté dans l'état de l'art, le modèle de contrats de *ConFract* [Collet *et al.* 2005] introduit trois types de contrats – des contrats d'*interface*, de *composition interne* et *externe* – pour prendre en compte les différentes caractéristiques architecturales des composants hiérarchiques. Le listing 6.1 présente une spécification externe exprimée dans le langage d'assertions exécutables *CCL-J*. Cette spécification externe est définie sur le composant `<ic>` présenté précédemment (cf. Fig. 6.1). Elle exprime un invariant de configuration, vérifié en fin de phase de configuration des composants, qui définit une contrainte de seuil minimal sur le nombre d'utilisateurs maximal. Le composant `<ic>` doit supporter un nombre maximal d'utilisateurs d'au moins 250.

```
on <ic>
 param max-users=250
 inv <ig>.attributes.getMaxUsers() >= max-users
```

Listing 6.1 – Spécification interne sur l'assemblage du composant `<ic>`.

Le listing 6.2 présente cette fois-ci une spécification interne définie sur le composant `<vs>` (cf. Fig. 6.2). Elle décrit une précondition sur la méthode `start` de l'interface `v` qui permet de vérifier, à l'entrée de cette méthode, que l'utilisation du CPU, estimé à partir de certains paramètres (flux des données multimédias, frameRate, nombre de clients du service), ne dépasse 60 %. La postcondition permet de vérifier qu'en sortie de la méthode, la vidéo est bien démarrée.

```
on <vs>
 context void v.start()
  pre c.expectedCPUUsage(getUrl().getDataSource(),
                         <this>.attributes.getFrameRate(),
                         <this>.attributes.getNbClients()) <= 60 //in %

  post v.isStarted();
```

Listing 6.2 – Spécification externe sur l'usage du composant `<vs>`.

Les contrats construits par le système *ConFract* regroupent les spécifications suivant leur catégorie (pré, post, invariant) et chaque clause de spécification devient une clause du contrat qui contient la spécification, mais aussi le contexte spatio-temporel d'interception ainsi que les références au contexte (composants, interfaces, etc.), nécessaires pour être évaluées. Par ailleurs, à chaque type de spécification (pré, postcondition, etc.), le modèle de contrats attribue aussi différents types de responsabilités aux composants impliqués [6]. Ces responsabilités peuvent ainsi être : *i)* des *garants*, représentant les composants qui s'efforcent de rendre la clause vraie et qui ont potentiellement la capacité d'agir pour traiter ces échecs, *ii)* des *bénéficiaires*, représentant les composants qui veulent pouvoir compter sur la véracité de la clause et qui peuvent être prévenus en cas de changement d'état de celle-ci (invalidation ou rétablissement), et *iii)* d'éventuels contributeurs, qui sont des composants nécessaires à l'évaluation de cette dernière.

Pour gérer les contrats, des *gestionnaires de contrats* sont introduits et placés sur la membrane de chaque composant composite. Ils sont implémentés dans la plate-forme *Fractal* par un nouveau contrôleur de contrats dédié. Chaque gestionnaire de contrats prend en charge le cycle de vie et l'évaluation *i)* du contrat de composition interne du composite sur lequel il est placé, *ii)* du contrat de composition externe de chacun des sous-composants, et *iii)* du contrat d'interface de chaque connexion dans son contenu. Le gestionnaire de contrats construit les contrats en fonction des événements d'assemblage qui surviennent sur les composants (insertion d'un sous-composant, connexion de deux interfaces, etc.), et met aussi à jour les contrats impactés lors des reconfigurations dynamiques sur l'architecture.

La figure 6.3 présente une vision textuelle du contrat construit à partir de la spécification interne du composant `<ic>` (cf. List. 6.1), la portée de ce contrat et l'élément qu'il référence (interface d'attributs qui étend `AttributeController`). Le contrat est géré par le gestionnaire de contrat du composite `<ic>`. Les responsabilités de cette clause sont le composant `<ic>` à la fois en tant que garant et bénéficiaire. Ce contrat définit une contrainte d'assemblage de ce composite vis-à-vis de son sous-composant `<ig>`. Le composite `<ic>` est alors seul responsable de son implémentation ; il garantit son assemblage et bénéficie de la justesse de celui-ci.

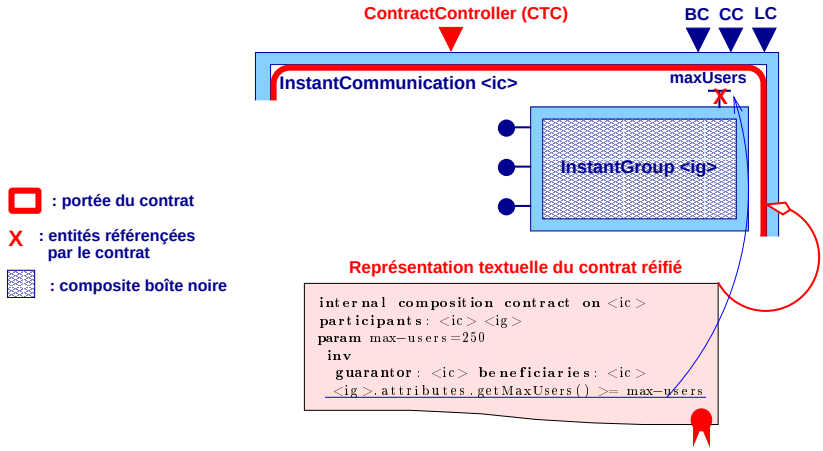


FIGURE *6.3 – Contrat interne sur l'assemblage du composant <ic>*

De même, la figure 6.4 présente une vision textuelle du contrat de composition externe construit à partir de la spécification externe (cf. List. 6.2), sa portée, et les éléments qu'il référence (interfaces fournie `v :Video` et requise `c :Config`). Le contrat est géré par le gestionnaire de contrats du composite

6. Les responsabilités associées à chaque type de spécification et de contrats sont précisément décrites dans l'annexe B.

<ia>. Les responsabilités de la précondition sont <ia> en tant que garant, <vs> en tant que bénéficiaire et <sc> en tant que contributeur. Il est à noter que le garant de cette précondition est bien l'englobant <ia>, et non pas <sf> qui agit en tant que simple utilisateur de l'interface, car c'est à <ia> que revient la responsabilité d'assurer la bonne utilisation générale de son sous-composant <vs> vis-à-vis des autres. <vs> quant à lui bénéficie de la véracité de la clause pour réaliser son service. Pour la postcondition, le garant est <vs>, et les bénéficiaires sont <sf>, connecté et client du service, et <ia> qui contient <vs>.

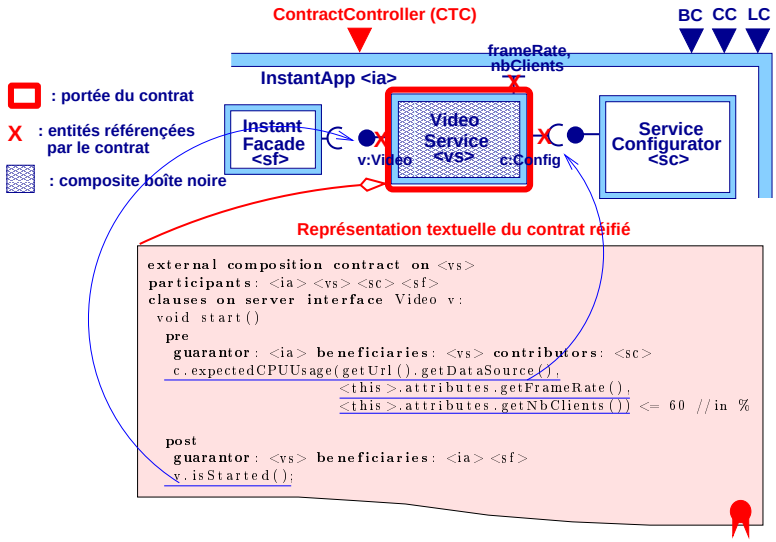


FIGURE 6.4 – *Contrat externe sur l'usage du composant <vs>*

Dans sa première version basée sur des assertions exécutables, le système *ConFract* effectue les vérifications des clauses à l'exécution des composants car les assertions peuvent être paramétrées par des éléments fonctionnels d'exécution (méthodes, valeurs effectives des paramètres). De plus, l'évaluation des contrats s'effectue selon certaines règles. Lorsqu'une méthode est appelée sur une interface *Fractal*, les préconditions du contrat d'interface sont d'abord évaluées, puis celles du contrat de composition externe du composant recevant l'appel, et enfin celles du contrat de composition interne. Des vérifications, dans l'ordre inverse, sont effectuées au retour de la méthode pour les postconditions et les invariants. Dans les cas où la vérification d'une clause n'est pas satisfaite, le gestionnaire du contrat en question lance une exception qui décrit tout le contexte de la violation. Dans nos exemples précédents, la clause exprimant l'invariant de configuration est violée pour un composant <ig> qui ne supporterait qu'une charge maximale de 100 utilisateurs (`maxUsers` vaut 100 par exemple alors que la contrainte spécifie un minimum de 250). La précondition du contrat externe, quant à elle, peut être violée lorsque l'utilisation estimée du CPU par le service de diffusion vidéo dépasse les 60% (`expectedCPUUsage` rend 80% par exemple), et la postcondition est violée si la vidéo n'est pas dans l'état démarré.

6.1.2 Principes du modèle

Comme les contrats sont découpés en clauses, le modèle de négociation définit et s'organise autour de *négociations atomiques*. Une négociation atomique est déclenchée pour chaque clause violée. Elle regroupe les différents éléments de base nécessaires à toute négociation et a pour objectif de rétablir la validité d'une clause en échec d'un contrat. De plus, comme les mécanismes proposés visent à conférer

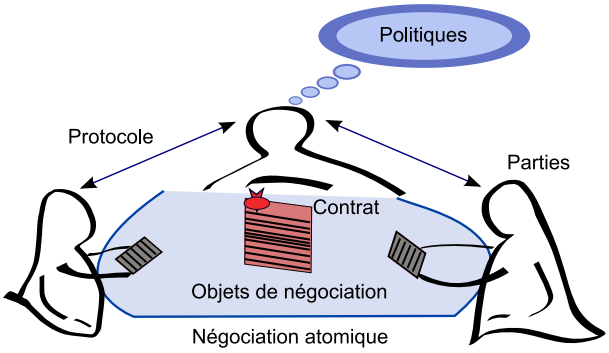


FIGURE 6.5 – *Organisation d'une négociation atomique*

davantage d'autonomie aux composants, les négociations atomiques vont faire interagir des *parties négociantes* selon un processus décentralisé, et chaque négociation atomique réunit donc des *parties* bien identifiées. Ces parties interagissent pour négocier autour d'*objets de négociation* que sont les clauses des contrats, à partir de leur rôle ou responsabilité dans celles-ci. De plus, pour conduire les interactions des parties, celles-ci sont organisées selon un *protocole* de négociation. Enfin, différentes *politiques* de négociation qui peuvent exploiter des informations fines contenues dans les contrats comme les responsabilités des composants, sont aussi intégrées (cf. Fig. 6.5).

La figure 6.6 décrit le schéma d'intégration général de la négociation dans le processus de contractualisation des composants. Les négociations atomiques sont attachées aux vérifications en échec des clauses. Celles-ci peuvent intervenir, juste après la construction des contrats lors de la phase de configuration des composants, pour effectuer des vérifications statiques sur les clauses, ou alors, lors de l'exécution des composants, lorsque les clauses sont vérifiées dynamiquement. Dans les deux cas, le processus général est identique. Les négociations atomiques sont déclenchées lorsqu'une clause est en échec, et elles peuvent notamment conduire à modifier les clauses négociées. Lorsqu'elles les négociations ne permettent pas de rétablir des clauses valides, elles sont en échec. Le contrat est alors rompu, et il est possible, en dehors de tout processus de négociation, de mettre en œuvre de façon spécifique divers actions de reconfiguration dynamique.

6.2 Objets de la négociation

Les objets de la négociation représentent les termes ou « sujets de discussion » pour lesquels une négociation est amenée à démarrer. Leur définition précise est importante pour toute négociation en général, car ces objets représentent les éléments communs qui font que des parties bien précises sont réunies et sont amenées à interagir au cours d'une négociation. Dans notre problématique, comme le modèle de négociation défini vise des plates-formes à composants contractualisés, en s'attachant à rétablir des clauses valides de contrats, les objets de nos négociations atomiques sont naturellement identifiés, et sont justement ces clauses en échec des contrats. Dans nos exemples précédents, ce sont ainsi les clauses qui spécifient l'estimation de l'utilisation CPU du service vidéo ou le seuil minimal d'utilisateurs maximal qui sont les objets de la négociation. Compte tenu de notre objectif de proposer un modèle pour conduire l'auto-adaptation par rétablissement de contrats valides, ces objets de la négociation sont les seuls éléments fixes du modèle de négociation. Les autres aspects du modèle (parties négociantes effectives, interactions, propositions et actions de négociation) sont précisés dans la suite.

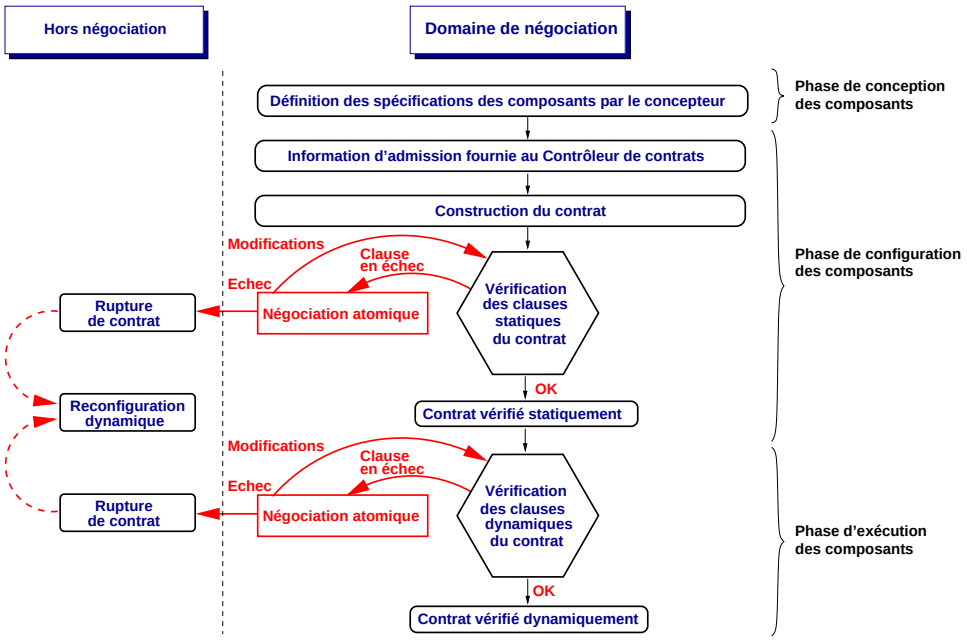


FIGURE 6.6 – *Intégration de la négociation dans le processus de contractualisation*

6.3 Parties

De façon spécifique à chaque négociation atomique, le modèle de négociation définit les parties négociantes suivantes : *(i)* le gestionnaire du contrat concerné, *(ii)* les entités participantes récupérées de la clause négociée avec leur responsabilités (bénéficiaire, garant, contributeurs) entièrement déterminées par le modèle de contrats, et *(iii)* un *participant externe*, introduit afin de représenter les intérêts d'une entité externe possédant une capacité de décision supérieure. Le gestionnaire de contrats a pour rôle de prendre en charge la gestion du cycle de vie des contrats et la réalisation de leurs vérifications ; comme la négociation fait partie intégrante de la contractualisation, son champ d'action est naturellement étendu pour qu'il prenne aussi en charge la négociation des contrats violés. Toutefois, comme ce dernier ne sait pas *a priori* rétablir seul les clauses en échec, il consulte les composants participants qui, compte tenu de leurs responsabilités dans la clause négociée, possèdent eux les diverses capacités pour rétablir la clause. Enfin, le participant externe est défini pour représenter toute partie externe destinée à définir ou modifier certains aspects du déroulement de la négociation. Ce participant peut être, par exemple, l'administrateur de l'application qui souhaite injecter des contraintes globales sur le système telles que des contraintes de déploiement, ou des données pour paramétrer le processus de négociation, comme par exemple, la négociabilité d'une clause étant donné le contexte de déploiement, une durée limite de la négociation ou encore la diffusion d'informations sur des négociations à d'autres niveaux de la hiérarchie des composants. Ce participant externe sert ainsi à intégrer divers éléments, vraisemblablement de haut-niveau (informations, contraintes, etc.), pour piloter la négociation. Toutes ces parties interagissent selon un protocole de négociation piloté par le gestionnaire de contrats. Ce dernier peut alors tirer parti des diverses responsabilités des parties impliquées et orienter le déroulement de la négociation selon différentes politiques.

Les parties négociantes des clauses de contrats des exemples précédents (cf. Fig 6.3 et 6.4) sont récapitulées dans la table 6.1 qui suit.

Clause concernée	Parties négociantes
Invariant du contrat interne	gestionnaire de contrats de <ic>, <ic> (garant), <ic> (bénéficiaire)
Précondition du contrat externe	gestionnaire de contrats de <ia>, <ia> (garant), <vs> et <sc> (bénéficiaire)
Postcondition du contrat externe	gestionnaire de contrats de <ia>, <vs> (garant), <ia> et sf (bénéficiaires)

TABLE *6.1 – Illustration des parties négociantes*

6.4 Protocole

À partir des clauses, l'ensemble des parties qui vont interagir lors des négociations atomiques sont clairement connues. Un protocole de négociation est alors nécessaire pour spécifier la dynamique des interactions entre les parties, et sa donnée induit très souvent une (nouvelle) organisation des parties en leur affectant des rôles spécifiques qu'ils doivent tenir au cours de la négociation.

Le protocole de négociation actuellement employé pour les négociations atomiques est adapté du *Contract-Net Protocol* (CNP) étendu [FIPA TC Communication 2002b], exploité pour de nombreuses applications (cf. page 53 pour plus de détails). Ce protocole propose de structurer les différentes parties d'une négociation selon une hiérarchie formée d'un initiateur et de participants (cardinalité 1-N), et organise simplement les échanges d'informations, entre l'initiateur et les participants, en trois phases selon un schéma général de type appel d'offres (appel d'offres, réception et analyse des offres, contractualisation).

Dans le système *ConFract*, le gestionnaire de contrats et les participants d'une clause sont directement liés par un niveau de hiérarchie dans une relation englobant/sous-composants directs (cf. B.3), l'utilisation du CNP se justifie ici car l'organisation des parties négociantes se projettent naturellement dans le protocole CNP, de la façon suivante. L'initiateur de la négociation est le gestionnaire de contrats situé au niveau du composant englobant. Il pilote le déroulement de celui-ci et et consulte les autres participants. Les participants sont les composants responsables, situés soit au même niveau que le gestionnaire de contrats, soit au niveau des sous-composants directs, et le participant externe. De cette façon, le gestionnaire de contrats, initiateur de la négociation, dispose d'une vision et de la portée d'action sur l'ensemble des éléments référencés (contrats, composants, etc.) Ils sont consultés par le gestionnaire de contrats pour des propositions visant rétablir la clause en échec. Le protocole de négociation atomique s'organise alors selon ces trois étapes :

1. l'initiateur *demande des modifications* aux parties sur la clause en échec,
2. les parties *soumettent des modifications*,
3. l'initiateur effectue les modifications proposées et revérifie la clause pour évaluer si les propositions suffisent à revalider la clause.

Ces étapes sont réitérées et la négociation se termine alors, soit lorsqu'une solution satisfaisante est trouvée, soit lorsqu'aucune modification ne permet de rétablir la validité de la clause en échec. En pratique, pour ne pas prolonger indéfiniment le mécanisme de rattrapage, le processus de négociation est aussi borné par une durée limite au delà de laquelle une négociation atomique est arrêtée. Si la clause est revalidée alors la négociation est finalisée et le cycle de vie normal du système reprend là où elle s'était arrêtée. Si la clause reste en échec, la négociation est alors *en échec* pour cette négociation atomique bien

précise. Il est alors possible, d'après d'autres règles de pilotage de haut-niveau, de modifier certains éléments de la négociation atomique et de relancer cette dernière avec de nouveaux éléments (exploitation d'autres politiques, paramétrages différents du participant externe, changement de protocole de négociation, etc.). Si les nouvelles tentatives de négociation atomique restent insuffisantes et ne permettent toujours pas de revalider la clause, alors la clause du contrat est *rompue*. Le processus de négociation se termine, et fait remonter cette information *d'échec de négociation*. D'autres actions de rattrapage, effectuées à un niveau plus haut, et portant sur des éléments de grains plus importants (code d'adaptation dynamique, reconfigurations de l'architecture des composants), pourraient alors être déclenchées, en dehors du processus de négociation (cf. section Discussion page 95).

6.5 Politiques

Une fois les parties et le protocole de négociation déterminés, différentes politiques peuvent être appliquées pour offrir diverses formes de négociation. Les politiques de négociation proposées exploitent les informations très fines des responsabilités des clauses de contrats, et elles déterminent le processus de négociation dès son initialisation. Ainsi, à partir des responsabilités des participants, deux politiques sont actuellement proposées et couvrent les principales responsabilités, bénéficiaire et garant, attribuées par le modèle de contrats aux composants impliqués. Une première politique dite par *concession* oriente la négociation vers les composants bénéficiaires des contrats. Elle reste au même niveau de hiérarchie concerné par la violation de contrats, et conduit à effectuer des *concessions* sur la contrainte spécifiée en modifiant soit entièrement la clause négociée, soit certains de ses termes. Une seconde politique dite par *effort* oriente, cette fois-ci, la négociation vers l'unique composant garant des contrats. Elle exploite les capacités du garant soit à faire des *efforts* pour rétablir la clause à son niveau, soit à *propager* la négociation dans sa composition afin de consulter de nouveaux composants susceptibles d'effectuer, à leur niveau, d'autres efforts qui contribueraient à revalider la clause mise en échec, au niveau du composant garant. Ces deux politiques couvrent les deux différentes responsabilités exploitables [7]. De plus, elles diffèrent radicalement l'une de l'autre en termes de niveau de hiérarchie des composants consultés ainsi que de propositions d'actions réalisables. Elles enrichissent ainsi le modèle de différentes formes de négociation qui peuvent être utilisées pour revalider des clauses de contrats. La suite détaille ces deux politiques.

6.5.1 Politique par concession

Principes

La politique par concession modifie l'initialisation de la négociation en faisant intervenir uniquement l'initiateur et les bénéficiaires en tant que parties négociantes. L'initiateur s'oriente alors vers les bénéficiaires de la clause et exploite leurs capacités à faire des *concessions*. Ainsi, l'initiateur effectue des demandes de concession aux bénéficiaires. Compte tenu de leur responsabilité, comme les bénéficiaires s'appuient sur une clause et expriment des attentes vis-à-vis de celle-ci, ces demandes de concessions ont bien un sens, car elles visent à exploiter les capacités potentielles des bénéficiaires à assouplir leurs attentes, s'ils y sont disposés. Les bénéficiaires peuvent alors, en fonction de leur contribution ou non à la clause en échec, relâcher les contraintes de la clause en échec *(i)* par *changement complet de la clause*

7. La responsabilité de contributeur n'est pas exploitée par le modèle de négociation car celle-ci décrit davantage des composants qui interviennent plus indirectement dans les clauses en étant par exemple nécessaires à leur fermeture (voir annexe B sur le cycle de vie des contrats dans *ConFract*). Ces composants contributeurs n'ont alors pas de rôle actif à jouer dans la négociation, contrairement aux composants bénéficiaires et garant dont les capacités, en terme de bénéfice/garantie vis-à-vis d'une clause, peuvent être exploitées.

pour une nouvelle clause moins contrainte (relâchement de la clause entière), ou par *changement de certains termes paramétrés de la clause* (relâchement de certains termes de la clause) [8]. Il est à noter que le fait que les concessions proposées soient plus ou moins contraints les uns par rapport aux autres n'est pas forcément mesurable. Cela dépend beaucoup du formalisme de spécification utilisé et de l'espace dans lequel les valeurs sont définies. Par exemple, en *QML*, comme les spécifications expriment souvent des valeurs de seuils sur des valeurs entières, il est possible d'évaluer, statiquement sur les formules, qu'une spécification est plus ou moins contrainte qu'une autre. Par contre, dans le langage *CCL-J*, comme il est possible de spécifier dans le cas général de nombreux éléments à l'exécution, la comparaison de deux spécifications exprimées dans ce formalisme n'est pas forcément possible.

La mise en œuvre de cette politique exploite fortement la distinction affinée entre les rôles de bénéficiaire principal et de bénéficiaire secondaire (cf. section B.3). En effet, en plus de la distinction due à la différence de visibilité, les bénéficiaires principaux sont ici distingués des bénéficiaires secondaires car ils possèdent des capacités de négociation plus développées. Les bénéficiaires principaux sont directement concernés par la clause et ont la capacité de modifier la clause. En revanche, les bénéficiaires secondaires ont un rôle plus passif et ne peuvent pas faire avancer la négociation. Les bénéficiaires principaux négocient donc en leur nom, ainsi qu'en celui des bénéficiaires secondaires. Dans notre exemple précédent, pour la postcondition du contrat de composition externe sur `<vs>` (cf. page 75), `<ia>` est le bénéficiaire principal car responsable de la correcte utilisation du contrat et de l'usage de `<vs>`, et `<sf>` bénéficiaire secondaire, car simple client du service et plus indirectement concerné.

Processus général

En cas d'échec d'une vérification, le processus de négociation piloté par la politique par concession se décompose selon les trois phases décrites dans la Fig. 6.7.

1. Tout d'abord (phase 1), l'initiateur demande la négociabilité de la clause en échec à tous les bénéficiaires, principaux et secondaires, et détermine la négociabilité de la clause par le biais d'une fonction additive pondérée. En effet, comme les avis exprimés par les différents bénéficiaires n'ont pas la même pertinence, différents poids sont attribués aux composants bénéficiaires, pour prendre en compte ces différences. Couplé à cela, la fonction additive pondérée permet alors à l'initiateur d'évaluer la négociabilité globale, qui se traduit par un seuil de négociabilité, en modérant l'avis exprimé par chaque composant par son poids. Ainsi, une clause est négociable *in fine* si un certain *seuil* de composants *importants* l'acceptent. Après évaluation de la négociabilité par l'initiateur, si la clause n'est pas négociable, la négociation atomique échoue pour cette politique.
2. Dans le cas contraire, si la clause est négociable (phase 2), l'initiateur demande aux bénéficiaires principaux de *faire des concessions* en lui passant une référence de la clause en échec. Les bénéficiaires principaux peuvent alors *proposer des concessions* qui représentent des modifications soit sur la clause entière, soit sur certains termes paramétrées de la clause. Ces étapes de demandes et propositions de modifications sont itérées, et pour chaque modification proposée, l'initiateur effectue les modifications puis réévalue la clause concernée. Si l'évaluation de la clause reste fausse alors l'initiateur annule la modification pour rétablir le contexte initial et redemande des concessions. Au contraire, si l'évaluation est vraie alors la négociation atomique est réussie, et l'initiateur finalise la négociation en demandant aux bénéficiaires d'effectuer les concessions correspondantes.
3. Autrement, dans une dernière étape (phase 3), dans le cas où aucune proposition ne s'avère être satisfaisante ou si l'initiateur ne reçoit que des demandes de retrait venant des bénéficiaires prin-

8. Dans ce cas, le bénéficiaire ne voit que la clause en échec, et selon l'expressivité du langage de spécification, il relâche la contrainte soit en modifiant certains termes donnés de la clause, soit en remplaçant entièrement la clause.

cipaux, il procède alors à une dernière consultation pour demander aux bénéficiaires principaux et secondaires l'autorisation de retirer la clause.

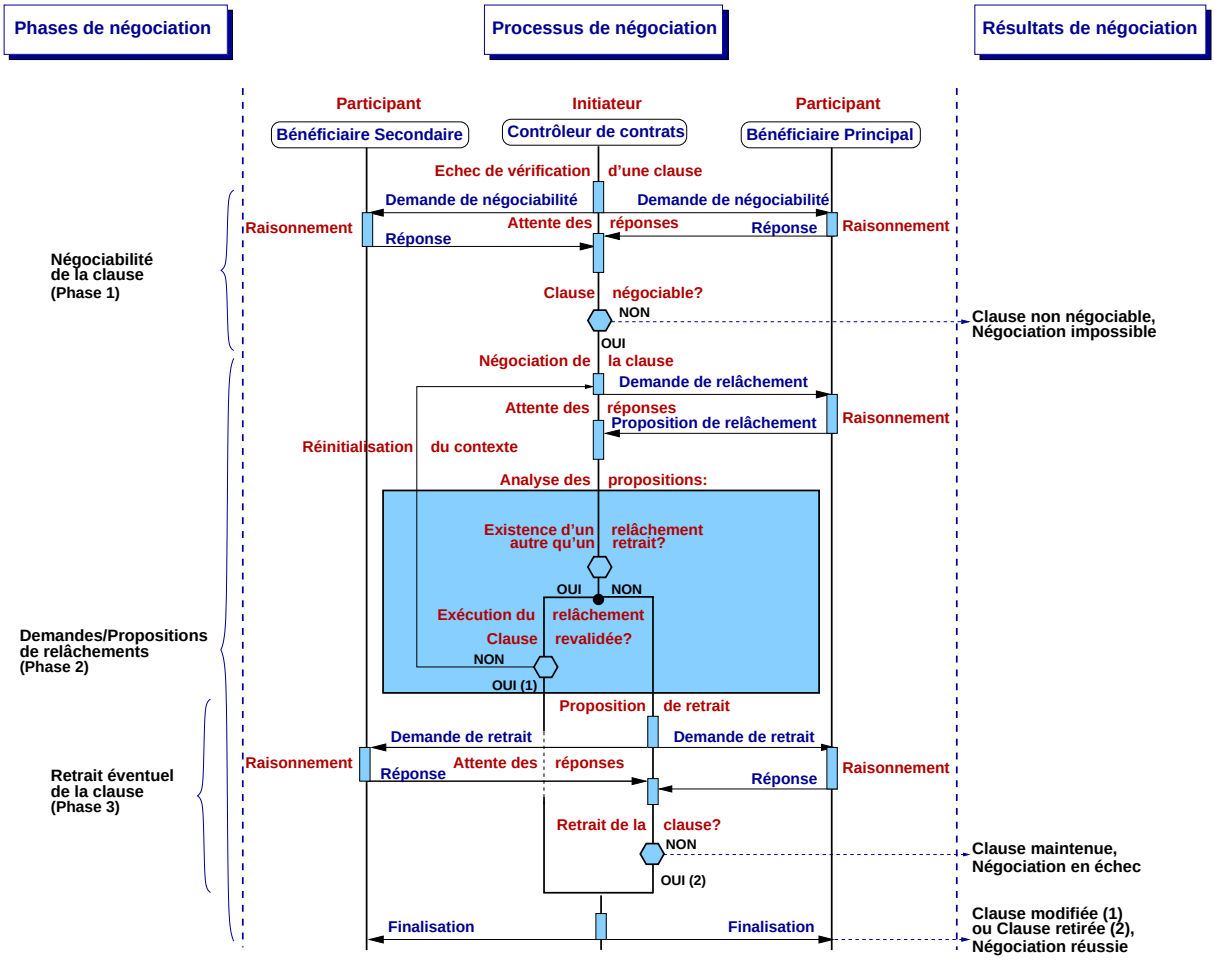


FIGURE 6.7 – *Processus de négociation atomique par concession*

Modèles de décision

Dans le déroulement la négociation par concession, les bénéficiaires principaux raisonnent en s'appuyant sur une liste d'alternatives qui décrit leurs préférences et représente les concessions successifs à faire au cours de la négociation. Ainsi, chaque composant C_i du système qui possède une responsabilité de bénéficiaire dans une clause identifiée id, associe à cette dernière un ensemble d'alternatives ordonnées, représenté de la façon suivante :

$$\mathcal{A}_{id,C_i} := \{A^1_{id,C_i}, A^2_{id,C_i}, ..., A^n_{id,C_i}, \text{STOP ou RELEASE}\}.$$

et dans laquelle chaque $A^j_{id,C_i},\ j \in [1,n]$ représente la j^{ieme} alternative (nouvelle formule, modifications d'un terme de la formule) du composant bénéficiaire C_i pour la clause id.

Ainsi, au cours de la négociation, à chaque fois que l'initiateur émet une demande de concession au composant, celui-ci renvoie successivement, à partir de cette liste d'alternatives, une concession qui

correspond à une des modifications de la clause ou des attributs. L'alternative STOP, resp. RELEASE, permet de notifier la fin de la concession en conservant la clause, resp. en autorisant le retrait de la clause. Les alternatives A^j_{id,C_i} sont optionnelles car un composant peut ne pas avoir de modifications à proposer sur la clause, en revanche STOP et RELEASE, sont obligatoires et exclusives car, au minimum, un bénéficiaire, doit exprimer si la clause sur laquelle il s'appuie doit être maintenue, ou au contraire peut être complètement retirée.

D'un point de vue méthodologique, ces modèles de décision doivent être fournis pour tout composant contractualisé qui a une responsabilité dans une clause dont on souhaite qu'elle puisse être négociée. Pour un système à composants contractualisés donné, les clauses et les contrats sont en nombre relativement réduit. De plus, comme parmi celles-ci, un certain nombre de clauses sont définies pour vérifier et détecter des erreurs fonctionnelles ou d'assemblages, elles ne sont pas destinées à être négociées. Il n'y a donc finalement qu'un très faible nombre de clauses négociables et pour lesquelles il faut fournir de telles listes d'alternatives, mais celles-ci sont malgré tout pertinentes compte tenu des propriétés qu'elles spécifient.

Bilan

La politique par concession exploite les capacités des composants bénéficiaires et conduit à effectuer des modifications sur la clause en échec (remplacement de la clause, modifications de termes de la clause, retrait/maintien de la clause), sur les attributs des composants dans le but de relâcher les contraintes ou leurs contributions à la clause. En revanche, avec cette politique, la négociation reste confinée à un seul niveau de la hiérarchie des composants et n'exploite pas réellement la notion de composant hiérarchique mise en avant par le modèle *Fractal*. Pour pallier ceci et fournir au modèle de négociation d'autres formes de négociation, nous intégrons au modèle une autre politique par effort qui exploite d'autres responsabilités et permet de propager cette négociation vers des composants de niveaux inférieurs.

6.5.2 Politique par effort

Principes

A la différence de la politique par concession, la politique de négociation dite par *effort* part du principe que l'initiateur de la négociation consulte, cette fois-ci, le garant de la clause en échec dans le but notamment d'exploiter sa capacité à re-satisfaire la clause et, à propager la négociation dans la hiérarchie. Ainsi, alors que dans la politique par concession, les bénéficiaires sont consultés pour qu'ils proposent des concessions visant à relâcher la clause sur laquelle ils souhaitent s'appuyer, dans la politique par effort, le composant garant est maintenant consulté pour qu'il propose des efforts visant à rétablir la clause qu'il doit satisfaire.

Ainsi, le composant garant peut avoir deux types de capacités. Il peut avoir des capacités *d'actions* propres qui lui permettent de prendre l'initiative de re-satisfaire la clause violée, et d'effectuer des actions d'effort à son niveau. Il peut aussi avoir des capacités de *propagation*, qui lui permettent de prendre en charge la négociation et consulter des sous-composants dans son intérieur pour exploiter et obtenir des propositions d'efforts de leur part. Dans ce cas, lors de la propagation, une nouvelle négociation atomique se reforme, et le garant et certains de ses sous-composants deviennent ainsi les nouvelles parties négociantes chargés d'interagir dans la négociation atomique.

Ces deux types de capacité sont découplés et l'ensemble des possibilités de la négociation par effort est donc couvert par la matrice des quatre cas issus du produit des ensembles {faire un effort à son niveau ou pas} X {propager ou pas}.

De façon plus précise, les capacités d'effort et de propagation du garant, induisent les deux classes d'actions suivantes :

- avec des capacités d'efforts : le composant garant *fait un effort* à son niveau de hiérarchie en proposant des *actions d'effort*. De telles actions sont complètement ouvertes mais sont régies par les deux contraintes suivantes : *(i)* les actions ne peuvent porter uniquement que sur des éléments dans la portée du composant garant (lui-même ou ses sous-composants s'il est composite), *(ii)* les actions ne doivent pas modifier les termes de la clause négociée, mais leurs valeurs peuvent l'être. Ainsi, ces actions d'efforts consistent, notamment, en des modifications de valeur de certains termes de la clause, ou alors des actions d'adaptations qui exécutent un code ouvert, dans la portée du garant et à clause constante.
- avec des capacités de propagation : il *sollicite la propagation* du processus de négociation, pour le moment confiné au niveau de hiérarchie de la violation, pour qu'il la prenne en charge et puisse consulter les composants dans son intérieur.
  Par ailleurs, comme en propageant la négociation, de nouvelles actions d'effort peuvent être proposées, cette fois-ci par les nouveaux composants à des niveaux de hiérarchie inférieurs, les contraintes de portée des actions, mentionnées précédemment, s'appliquent aussi à ces nouveaux composants, et ces derniers ne sont censés proposer que des efforts en relation avec la propagation effectuée (cf. éléments de raisonnements ci-après).

Du point de vue des possibilités de négociation offertes par le modèle, cette dernière caractéristique de propagation est essentielle, car elle vise, dans les cas favorables, à exploiter la composition hiérarchique du garant pour que celui-ci puisse rendre vraie une clause en échec dans laquelle il est garant. Une négociation propagée revient alors à consulter des sous-composants dans l'intérieur du composant garant afin que ceux-ci puissent effectuer, à d'autres niveaux inférieurs, des efforts qui permettent de revalider la clause en échec, au niveau du garant. Il est à noter que la responsabilité du garant est ici pleinement exploitée car c'est lui qui doit agir, en faisant un effort à son niveau ou en propageant la négociation, pour rendre vraie une clause de contrat. Au contraire, dans la politique par concession, les bénéficiaires ne voient une clause que dans l'optique de pouvoir s'y appuyer et sans qu'ils aient les moyens, ni de raisonner, ni d'agir pour la rendre vraie.

Pour réaliser cette politique par effort complètement, un verrou majeur réside dans la capacité du modèle à pouvoir propager avec précision la négociation. Le paragraphe suivant décrit les besoins du modèle et les besoins de raisonnement. Les mécanismes de propagation sont complètement décrits dans le chapitre 7 qui suit.

Eléments de raisonnement pour la propagation

Pour pouvoir propager la négociation dans la hiérarchie des composants et déterminer avec précision les nouveaux composants qui doivent intervenir, le modèle de négociation doit pouvoir s'appuyer sur divers éléments lui permettant de raisonner. Ces éléments de raisonnement doivent fournir les informations compositionnelles suivantes :

1. ils doivent permettre d'identifier clairement les nouveaux composants vers lesquels la négociation se propage,
2. ils doivent indiquer précisément comment la propriété spécifiée dans la clause négociée se décompose d'un niveau de hiérarchie de composants à l'autre. En effet, comme la négociation porte sur des clauses qui spécifient des propriétés extrafonctionnelles, et qu'une propriété peut s'exprimer, dans le cas le plus général, en fonction de plusieurs autres propriétés, les relations de décomposition des propriétés représentent alors ce fil conducteur qui permet de propager la négociation,
3. enfin, les liens entre les propriétés négociées et les composants qui participent à la négociation, doivent être clairement explicités. En particulier, comme la propagation va exploiter les relations

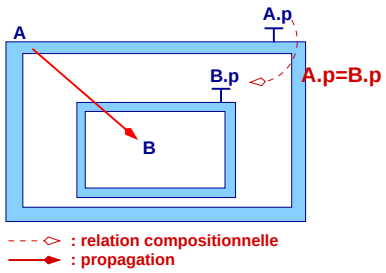
de décomposition des propriétés négociées, celles-ci doivent être modélisées autant que possible au niveau des composants eux-mêmes.

Dès lors que toutes ces informations compositionnelles sont réunies, le modèle de négociation peut alors piloter la propagation de la négociation. La décomposition de la propriété négociée est connue (point 2) ; les composants qui réalisent ces propriétés sont identifiés (point 3), et ces composants sont alors ceux qui sont consultés dans la politique par effort (point 1).

Pour cela, le modèle de négociation s'appuie sur une modélisation explicite des propriétés extrafonctionnelles ainsi que sur des relations compositionnelles associées aux propriétés négociées. Ces relations compositionnelles permettent justement d'indiquer très clairement à la fois la façon dont *se décompose* une propriété extrafonctionnelle dans une composition de composants, ainsi que la contribution de chaque composant à la réalisation de celle-ci. C'est alors à partir de la connaissance de cette contribution que les demandes d'efforts sont formulées.

Nous illustrons ces points au travers de deux exemples favorables dans lesquels les possibilités de raisonnement compositionnel sont simples :

Exemple 1. Considérons la propriété `A.p` d'un composite `A` qui contient un composant unique `B` avec une propriété `B.p` (cf. Fig. ci-après). La propriété `p` est clairement modélisée au niveau de chaque composant `A` et `B`, ces derniers sont précisément connus, et peuvent être consultés dans le cas d'une négociation sur `p`. De plus, supposons que la propriété exprimée au niveau de `A` soit exactement le reflet de la même propriété exprimée au niveau de `B`, alors la relation compositionnelle suivante `A.p=B.p` permet de connaître exactement à la fois comment se décompose la propriété `A.p` de `A` selon son intérieur (ici `B`) ainsi que la contribution exacte de `B` dans la propriété de `A` (ici il y a identité).

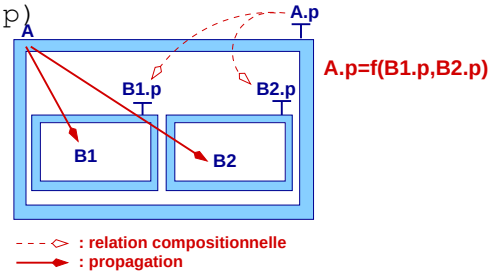


Ainsi, au niveau de `A`, la propriété est clairement modélisée (c'est `A.p`). Au niveau de `B`, il en est de même (c'est `B.p`), et la relation compositionnelle `A.p=B.p` exprime la décomposition de `p` du niveau de `A` au niveau de `B`, et permet d'orienter la négociation. Pour la négociation d'une clause qui porte sur la propriété `A.p` et pour laquelle `A` est garant, alors comme la réalisation de la propriété `A.p` de `A` est explicitée par la relation compositionnelle, il est possible d'orienter la négociation depuis `A` vers `B` pour consulter ce dernier compte tenu de sa contribution à la propriété. Cette propagation se fait ici naturellement, et est rendue possible, d'une part, par le fait que les propriétés `p` sur `A` et sur `B` soient clairement explicitées (sinon les propriétés ne peuvent pas être référencées, ou les composants porteurs être identifiés), et d'autre part, par le fait que la relation compositionnelle décrive vers qui (`B`), et comment (les deux propriétés sont égales) se réalise la propriété `p` de `A` dans son intérieur (sinon les nouveaux composants restent inconnus, ou leur contribution reste inconnue).

Il est à noter que, bien qu'étant simple, ce cas se produit couramment dans des situations réelles, lorsque des composants patrimoniaux sont réutilisés et enveloppés dans de nouveaux composites avec certains de leurs services conservés ou redéfinis, et certaines de leurs propriétés extrafonctionnelles (propriétés de qualité, caractéristiques des composants), conservées et externalisées au niveau de l'englobant.

Exemple 2. Dans un autre cas à peine plus complexe (c.f Fig. ci-dessous), la propriété `A.p` peut maintenant être liée à la même propriété, `B1.p` et `B2.p`, modélisées au niveau de ses sous-composants `B1` et `B2`.

Dans ce cas, la relation compositionnelle `A.p=f(B1.p,B2.p)` où `f` est une fonction quelconque (`f :=Min` ou `f` $:\Sigma$, pour prendre un exemple concret), exprime alors clairement la décomposition de `A.p` en fonction des propriétés `B1.p` et `B2.p` sur `B1` et `B2`. De même que précédemment, `p` est clairement modélisée au niveau de `A` puis au niveau des sous-composants `B1` et `B2`, et la relation compositionnelle `A.p=f(B1.p,B2.p)` décompose `p` du niveau de `A` au niveau de `B1` et `B2` (il y a conservation de la propriété mais décomposition sur plusieurs composants, avec la fonction arithmétique f).



Ainsi, la propagation d'une négociation qui porte sur une clause spécifiant `A.p`, et dans laquelle `A` est garant, conduirait à orienter la négociation vers `B1` et `B2` pour consulter ces derniers de sorte qu'ils puissent proposer des efforts, compte tenu de leur contribution à la propriété `A.p`. Cet exemple peut se généraliser au cas d'une formule compositionnelle qui lie une même propriété sur n composants, et de façon évidente, le cas le plus général consiste en une propriété qui se décompose selon plusieurs autres propriétés portées par différents composants (n propriétés sur m composants exprimée par une relation matricielle).

Dans nos exemples précédents, il se trouve que comme le composant `<ig>` effectue la gestion des fonctionnalités relatives aux utilisateurs et aux groupes, il est architecturé de la façon suivante (cf. Fig. 6.8), et contient notamment un sous-composant `UserManager` qui gère l'ensemble des fonctionnalités relatives aux utilisateurs connectés dans le système. La propriété `maxUsers` est donc effectivement implémentée par ce sous-composant `UserManager` et simplement externalisée au niveau du composite `<ig>`.

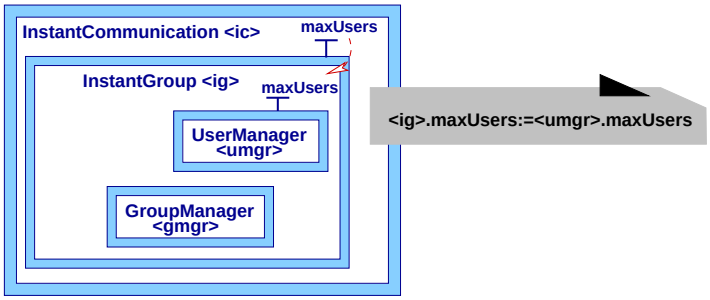


FIGURE 6.8 – *Exemple de relation compositionnelle sur le composant <ig>*

Ainsi, pour cette propriété, la formule compositionnelle suivante : <ig>.maxUsers := <umgr>.maxUsers permet alors de lier la même propriété `maxUsers` de `<ig>` à celle de son sous-composant `<umgr>` : la propriété `maxUsers` est réellement implémentée au niveau du composant `<umgr>` de gestion des utilisateurs et simplement reflétée au niveau de son englobant. La négociation se propage vers `<ig>` pour consultation de `<umgr>` contributeur à la propriété.

D'autres exemples de relations compositionnelles sont décrites dans l'annexe A.

Processus général

Comme nous l'avons précédemment indiqué (cf. section 6.5.2), l'ensemble des possibilités de la négociation par effort est couvert par les quatre cas issus du produit des ensembles {faire un effort à son

niveau ou pas} X {propager ou pas}. Le composant garant peut ainsi présenter les capacités suivantes :

- ne pas faire un effort à son niveau et ne pas propager ;
- faire un effort à son niveau et ne pas propager ;
- ne pas faire un effort à son niveau et propager ;
- faire un effort à son niveau et propager.

À partir de ces quatre capacités du garant, le processus de négociation par effort se décompose alors selon les phases suivantes :

1. Tout d'abord (phase 1), de la même façon que dans la politique par concession, l'initiateur demande la négociabilité au composant garant, et évalue la négociabilité de la clause (cf. Fig. 6.9). Si le composant garant n'a pas de capacité d'agir dans cette négociation – il *(ne peut ni faire un effort à son niveau* et *ni propager)* – alors la clause est pour lui non négociable, et il répond dans ce sens. Dans ce cas, comme le garant est unique et qu'il est le seul participant consulté dans cette politique, l'initiateur arrête la négociation atomique, et celle-ci échoue pour la politique par effort.

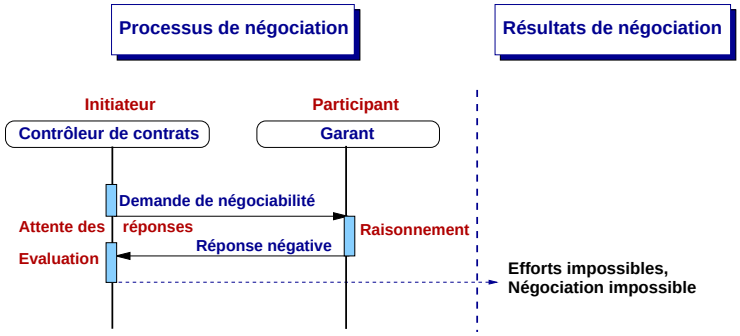


FIGURE 6.9 – *Phase de négociabilité*

2. Dans le cas contraire (phase 2), la clause est négociable et, le garant exprime par cela sa capacité à agir au cours de la négociation, en négociant ou en propageant celle-ci. L'initiateur demande alors au garant des *efforts* en lui passant une référence de la clause en échec. Par la suite, le garant peut avoir trois types de comportement, qui induisent trois formes de déroulement de la négociation. Il peut : *faire un effort à son niveau* et *ne pas propager*, ou *ne pas faire un effort à son niveau* et *propager* ou enfin *faire un effort à son niveau* et *propager*.
   a) Si le garant présente la capacité de (*faire un effort à son niveau* et *ne pas propager*), alors sa seule proposition d'effort consiste à exécuter des actions d'efforts conformes aux contraintes indiquées (cf. Principes) et qui permet de revalider la clause en échec (cf. Fig. 6.10). La garant propose ainsi à l'initiateur la réalisation d'une action qui consiste soit à modifier des termes de la clause, soit à réaliser une action d'adaptation.
   Si l'action consiste à modifier des termes de la clause, alors l'initiateur effectue les modifications, et revérifie la clause pour voir si elle est revalidée.
   Si l'action consiste en une action d'adaptation alors, comme le code de l'adaptation est privé au garant, l'initiateur ne reçoit du garant que l'information de la possibilité de réaliser ce code, et il doit alors simplement l'accepter ou refuser sans connaître la nature des actions ni leurs conséquences [9].

9. Une action d'adaptation est entièrement sous la responsabilité du garant qui ne sollicite de l'initiateur que son aval pour la réalisation. L'initiateur n'a pas à raisonner sur les actions proposées par les participants consultés. En revanche, comme il est responsable du contrôle et du pilotage de la négociation, il est, à ce titre, tenu d'accepter ou refuser qu'une action d'adaptation se fasse ou pas dans le cadre de la négociation atomique qu'il pilote.

Si l'initiateur refuse la réalisation de l'action d'adaptation alors il demande au garant de nouvelles actions d'effort. Si l'initiateur l'accepte, alors le garant effectue son action d'adaptation, et comme celle-ci conserve les éléments de la clause, l'initiateur revérifie la clause concernée pour évaluer si elle est rétablie.
Dans tous les cas, quelque soit l'action réalisée (modification de termes ou action d'adaptation), comme celle-ci préserve les termes de la clause (cf. Principes), l'initiateur opère une vérification de la clause pour évaluer le rétablissement de la clause négociée. Si la clause est rétablie, alors la négociation atomique se termine avec réussite, sinon l'initiateur demande au garant de nouvelles actions d'efforts. Si aucun des efforts proposés par le garant n'a permis de revalider la clause, alors la négociation se termine en échec.

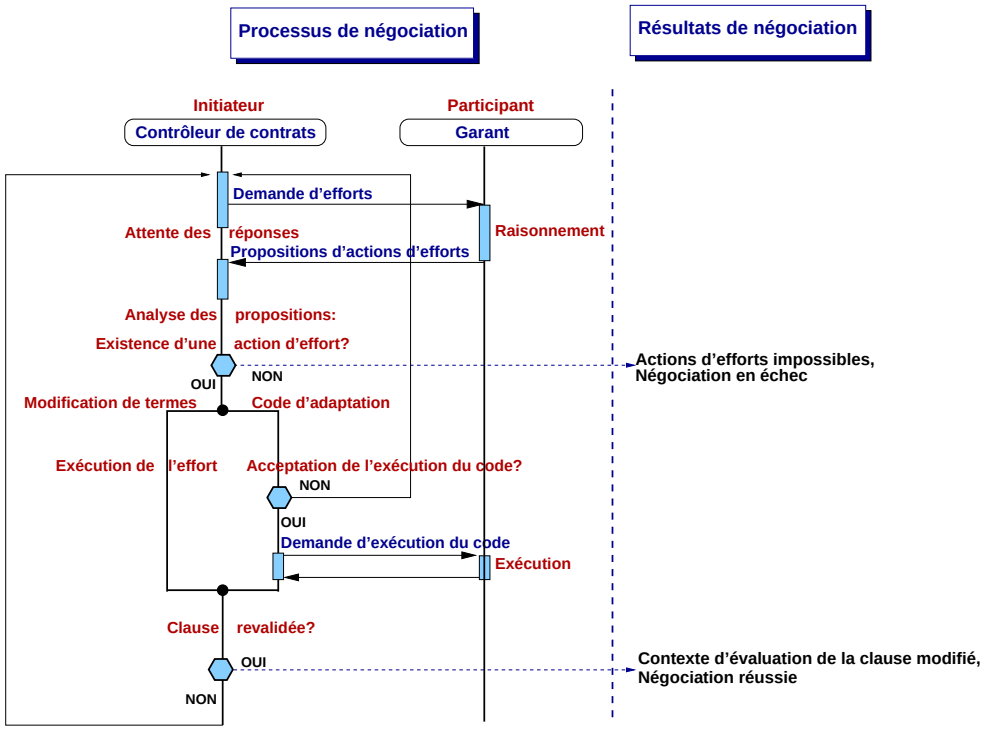


FIGURE *6.10 – Phase de propositions d'actions d'efforts*

b) Si le garant présente la capacité de (*ne pas faire un effort à son niveau* et *propager*), alors à la demande d'effort faite par l'initiateur, le garant répond en proposant de prendre en charge la négociation (cf. Fig. 6.11). Si l'initiateur refuse alors la négociation échoue pour cette politique. Si l'initiateur accepte la propagation, alors l'initiateur transmet au garant le contexte de la négociation, et c'est alors le gestionnaire de contrats du garant qui prend le relais et à qui revient le pilotage de la négociation dans son intérieur. Comme le garant a proposé la propagation de la négociation dans son intérieur, il a nécessairement une référence vers les sous-composants qu'il doit consulter [10]. Ainsi, le gestionnaire de contrats du garant consulte alors les sous-composants

10. Ces composants sont connus du garant soit parce qu'ils implémentent la propriété négociée (le garant est responsable de l'usage de ses sous-composant, la propriété négociée est réalisée à leur niveau, et le garant sait vers qui propager en connaissant le lien d'implémentation), soit parce qu'ils apparaissent parmi les contributeurs de la formule compositionnelle d'une propriété du garant (le garant est lui-même responsable de son implémentation, la propriété négociée est implémentée à son niveau, et le garant sait vers qui propager en connaissant la relation compositionnelle qui décompose la propriété à son niveau selon ses sous-composants).

clairement identifiés pour leur demander des efforts. Le processus de négociation par effort reprend alors à ce niveau de hiérarchie : le gestionnaire de contrats demande la négociabilité aux nouveaux participants (cf. phase 1), et ces derniers, s'ils veulent négocier, peuvent proposer des efforts qui consistent en la modification de la propriété propagée, en une action d'adaptation ou en une nouvelle propagation.

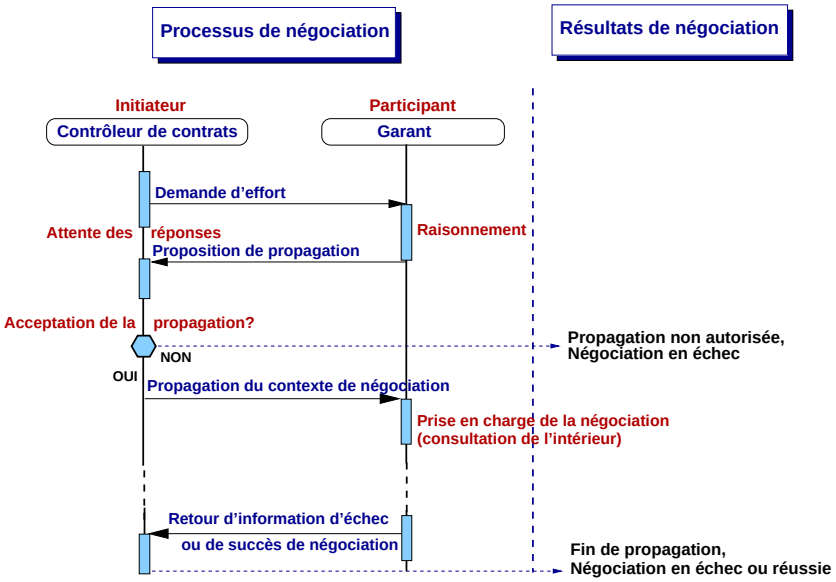


FIGURE *6.11 – Phase de proposition de propagation*

c) Si le garant présente la capacité de (*faire un effort à son niveau* et *propager*), alors il s'agit du cas où les capacités de négociation par effort du garant sont les plus riches. Ce cas regroupe les déroulements de négociation des deux cas précédents, et la négociation revient alors à coordonner les phases d'actions d'efforts et de propagation. Chaque garant définit alors un ordre de préférence par défaut (faire un effort d'abord ou propager d'abord) ou, s'ils sont équivalents pour lui, il en informe le gestionnaire de contrats qui choisit l'ordre d'exécution.

De la même façon que dans la politique par concession (cf. section 6.5.1.0), les mêmes règles permettant d'isoler le système du processus de négociation sont appliqués au démarrage de la négociation par effort.

Bilan

La politique par effort exploite les capacités de l'unique composant garant et conduit à effectuer soit des actions d'efforts contraintes (modifications de valeur de termes de la clause, code d'adaptation) pour directement re-satisfaire une clause, soit à propager la négociation dans l'intérieur du garant, pour consulter et exploiter ses sous-composants. La propagation de la négociation requiert, en particulier, de pouvoir s'appuyer sur divers éléments de raisonnements, comme par exemple, une modélisation explicite des propriétés extrafonctionnelles et des formules compositionnelles permettant de décrire la décomposition d'une propriété négociée dans une composition de composants. Le chapitre 7 qui suit décrit en détail les patrons d'intégration des propriétés extrafonctionnelles et le support proposé pour raisonner sur des propriétés compositionnelles.

6.5.3 Illustration des politiques

Cette sous-section illustre les principales caractéristiques des deux politiques présentées précédemment au travers des deux exemples de contrats présentés précédemment (cf. page 72) :

- Le premier exemple concerne deux clauses d'un contrat de composition externe exprimant une pré et une postcondition. Pour la première clause de ce contrat, la politique par concession conduit à modifier tout ou partie de la clause négociée, alors que la politique par effort effectue une modification sur un attribut du composant bénéficiaire. La seconde clause, quant à elle, présente rapidement un exemple d'échec de négociation par les deux politiques ;
- Le second exemple porte sur un contrat de composition interne. La spécification exprimée concerne un invariant de configuration. La politique par concession conduit à différents scénarii de négociation (refus de négociation, retrait total de la clause ou modification d'un terme paramétré), alors que la politique par effort se propage dans la hiérarchie du composant garant et conduit à effectuer une modification d'attribut d'un sous-composant contribuant à la propriété spécifiée dans la clause négociée.

Contrat de composition externe

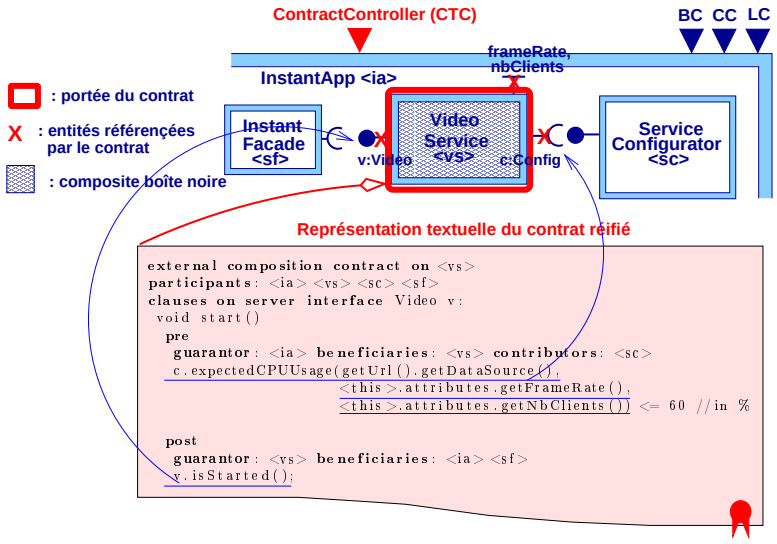


FIGURE 6.12 – *Rappel du contrat de composition externe sur <vs>*

- *Clause sur l'utilisation estimée du CPU :*
  La clause du contrat externe sur `<vs>` (cf. Fig. 6.12) vérifie à l'entrée de la méthode `start()` les bonnes conditions, en terme d'utilisation du CPU, pour la réalisation du service de diffusion vidéo, estimées à partir des valeurs de certains paramètres (flux multimédia courant, état des ressources CPU et nombre de clients). Cette clause est donc vérifiée à l'exécution du système et est violée, par exemple, si l'utilisation du CPU pour la diffusion vidéo est estimée à plus de 60%. Dans ce cas, les parties négociantes qui sont le gestionnaire de contrats de `<ia>` comme initiateur, le composant bénéficiaire `<vs>`, et le composant garant `<ia>`, interviennent dans une négociation atomique.

  Politique par concession. Dans la politique par concession, l'initiateur consulte `<vs>` et nous établissons un scénario dans lequel le composant `<vs>` propose une concession qui consiste à

changer entièrement la clause. Ainsi, l'initiateur demande tout d'abord la négociabilité. Comme <vs> a les moyens de négocier, il accepte la négociation et s'appuie sur la liste d'alternatives suivante, pour orienter ses concessions successifs : $\mathcal{A}_{pre,<pr>} := \{\Psi, \text{STOP}\}$ avec Ψ, décrivant la nouvelle clause :

$$\text{c.expectedCPUUsage(getURL().getDataSource(),}$$
$$\text{<this>.attributes.getFrameRate(), getNbClients())} \leq 80.$$

Cette nouvelle clause spécifie un seuil d'utilisation de CPU plus élevé ($80\,\%$ au lieu de $60\,\%$), et exprime ici la capacité de <vs> à s'appuyer sur une contrainte moins forte [11].
Ainsi, à la première demande de concession de l'initiateur, <vs> propose la nouvelle clause Ψ, et l'initiateur effectue le remplacement et la revérification de la clause. Si la clause est revalidée alors la négociation se termine avec un succès. Sinon, si la modification n'est pas suffisante et ne permet pas de rétablir une clause valide, alors l'initiateur annule la dernière modification et demande de nouvelles concessions. <vs> répond alors en proposant la fin des concessions et la négociation s'arrête avec maintien de la clause par l'alternative STOP. Comme <vs> est le seul bénéficiaire, la négociation par concession s'achève ainsi sur un échec avec le maintien de la clause. Par ailleurs, au lieu de proposer l'alternative STOP, le composant <vs> aurait aussi très bien pu proposer l'alternative RELEASE pour indiquer le retrait de la clause. Dans ce cas, comme la clause est entièrement retirée, la négociation s'achève avec succès, mais plus aucune vérification n'est effectuée pour détecter une sur-utilisation du CPU au démarrage du service de diffusion vidéo.

Politique par effort. Dans la politique par effort, l'initiateur consulte maintenant le garant <ia> (voir Fig. 6.13). Comme la propriété est implémentée au niveau du sous-composant <vs> de <ia>, alors <ia> accepte de négocier par effort et propose de propager la négociation (*étape 1*). Son gestionnaire de contrats prend alors en charge la négociation et consulte <vs> pour des demandes d'efforts, car ce dernier réalise la propriété frameRate qui intervient dans la clause négociée (*étape 2*). <vs> propose alors des modifications sur cette propriété, en s'appuyant sur sa liste d'alternatives associée $\mathcal{A}_{pre,<vs>} := \{(\text{framerate} \leftarrow \frac{\text{framerate}}{2}), \text{STOP}\}$.
Ainsi, à la première demande de concession de l'initiateur, <vs> propose d'ajuster son attribut `frameRate` avec l'alternative $\text{framerate} \leftarrow \frac{\text{framerate}}{2}$. L'initiateur effectue la modification et revérifie la clause. Si la clause est revalidée alors la négociation se termine avec un succès. Sinon, si la modification n'est pas suffisante et ne permet pas de rétablir une clause valide, alors l'initiateur annule la dernière modification et demande de nouveaux efforts. <vs> répond alors en proposant la fin de la négociation avec maintien de la clause par l'alternative STOP, car diminuer davantage le `frameRate` risquerait d'entraîner une diffusion saccadée de la vidéo. Comme <vs> est le seul composant qui contribue à la clause négociée, la négociation par effort s'achève ainsi sur un échec avec le maintien de la clause.
Il est à noter que le composants <vs> est consulté dans chacune des deux politiques. En revanche, son rôle n'est pas le même dans les deux cas : dans la politique par concession, il agit en tant que bénéficiaire direct de la clause qui peut proposer des modifications concernant la clause sur laquelle il s'appuie, alors que dans la politique par effort, il intervient après propagation et agit plus indirectement en proposant de reconfigurer son paramètre frameRate car il implémente le terme correspondant frameRate dans la clause négociée.

- *Clause sur l'état du service vidéo :*
  La clause qui vérifie l'état du service vidéo à la sortie de la méthode `start` est vérifiée à l'exécution du système et est violée, si à la fin de la méhode `start` la méthode le service vidéo n'est pas

11. Ici, cela est mesurable du fait que la nouvelle clause ne diffère de celle négociée que la valeur du seuil moins contraint.

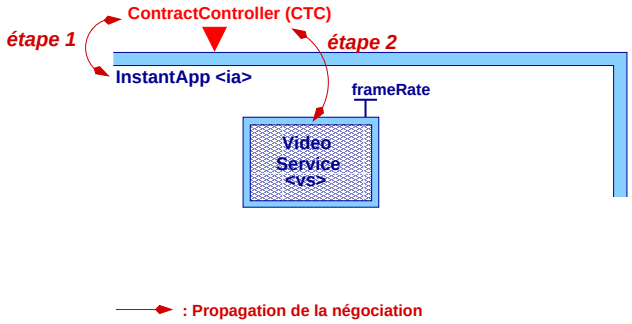


FIGURE 6.13 – *Schéma de propagation pour la propriété FrameRate*

dans l'état 'démarré' (`isStarted` rend faux). Dans ce cas, une négociation atomique démarre avec les parties négociantes suivantes : le gestionnaire de contrats de `<ia>` dans le rôle de l'initiateur consulte les composants bénéficiaires `<ia>` et `<sf>` dans la politique par concession, et le composant garant `<vs>` dans la politique par effort. Toutefois, comme cette violation traduit une erreur de fonctionnement du composant `<vs>`, nous considérons ici que la clause n'est pas destinée à être gérée par la négociation. Dans la politique par concession, le bénéficiaire principal accepte alors la négociation et propose l'alternative STOP pour demander le maintien de la clause (pas de concessions possibles et détection importante). Dans la politique par effort, le garant refuse directement la négociation car il est dans l'impossibilité de proposer des efforts pour revalider la clause (erreur vraisemblablement interne au composant `<vs>`). La négociation se termine en échec pour les deux politiques, et cet échec de négociation est ensuite signalée au plus haut-niveau.

Contrat de composition interne

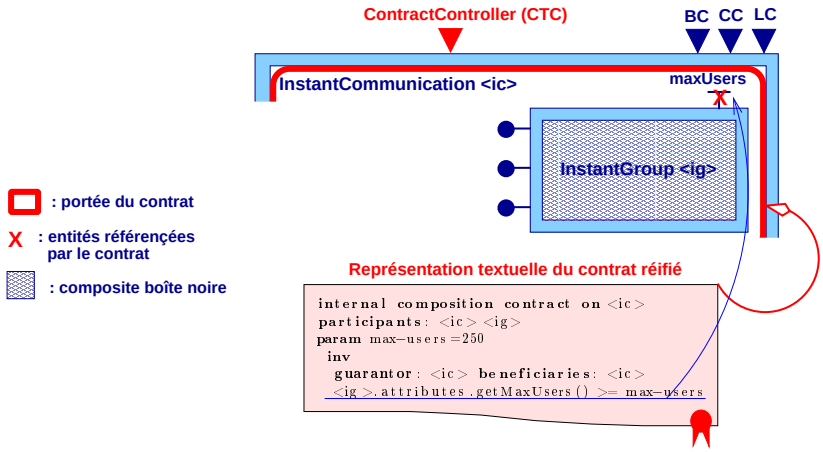


FIGURE 6.14 – *Rappel du contrat de composition interne sur <ic>*

- *Clause sur le seuil minimal d'utilisateurs maximal :*
  Cette clause du contrat de composition interne de `<ic>` (cf. Fig. 6.14) décrit certains aspects de son assemblage (seuil minimal du nombre maximal d'utilisateurs). L'invariant de configuration qui porte sur le seuil minimal d'utilisateurs maximal, est vérifié en fin de configuration et est violée, par exemple si le nombre maximal d'utilisateurs (`maxUsers`) effectifs vaut 100 alors que le seuil

minimal spécifié (`max-users`) vaut 250. Les parties impliquées dans la négociation atomique sont : le gestionnaire de contrats de <ic> dans le rôle de l'initiateur, et <ic> lui-même dans le rôle de composant à la fois garant et bénéficiaire.

Politique par concession. L'initiateur consulte tout d'abord <ic> en tant que bénéficiaire pour lui demander la négociabilité. Ce dernier accepte la négociation et peut proposer, suite à la demande de concession de l'initiateur, différentes concessions qui correspondent à plusieurs scénarii.

- avec la liste d'alternative $\mathcal{A}_{inv,<ic>} := \{\text{STOP}\}$, <ic> exprime avec STOP sa volonté de s'appuyer sur cette clause telle quelle. <ic> traduit ainsi le fait que le système doit absolument accepter un seuil minimal d'utilisateurs maximal de 250 utilisateurs. Dans ce cas, comme <ig> est l'unique bénéficiaire aucune autre concession ne peut être proposée et la négociation par concession se termine par un échec.
- avec la liste d'alternative $\mathcal{A}_{inv,<ic>} := \{\text{RELEASE}\}$,<ic> propose le retrait de la clause avec l'alternative RELEASE. Dans ce cas, il n'y a plus de contrainte de seuil minimal du nombre d'utilisateurs maximal, et en particulier un composant <ig> qui ne supporte qu'un nombre maximal de 2 utilisateurs, conviendrait alors parfaitement. Comme <ig> est l'unique bénéficiaire, l'initiateur effectue alors le retrait, et la négociation se termine avec succès.
- dans une situation intermédiaire, <ic> peut aussi proposer différents concessions successives sur le terme de la clause qui spécifie le paramètre maximal du nombre d'utilisateurs (`max-users`), avec la liste d'alternative suivante $\mathcal{A}_{inv,<ic>} := \{\text{max-users} \leftarrow 150, \text{max-users} \leftarrow 100, \text{max-users} \leftarrow 50, \text{STOP}\}$. <ic> propose alors successivement une concession qui consiste à modifier la valeur du paramètre `max-users` de la clause pour exprimer sa capacité à s'appuyer sur une clause moins contrainte. Ainsi, pour chaque alternative, l'initiateur effectue la modification de la clause et réévalue celle-ci avec le contexte initial de la violation. Dès qu'une alternative suffit à revalider la clause, alors la négociation s'arrête avec succès. Sinon, avec l'alternative STOP, <ic> propose la fin de la concession avec maintien de la clause. Ici, comme la clause est violée avec l'attribut `maxUsers`$= 100$, la première modification paramètre `max-users` qui suffit à rétablir une clause valide est max-users$\leftarrow 100$.

Politique par effort. Dans la politique par effort, le gestionnaire de contrats de <ic> consulte toujours <ic> mais cette fois-ci dans son rôle de garant. Nous décrivons ici un scénario dans lequel le composant <ic> exploite sa responsabilité de garant vis-à-vis de son assemblage et de l'usage de son sous-composant <ig> pour accepter la négociation et négocier par effort (voir Fig. 6.15). Ainsi, après la phase de consultation initiale pour la négociabilité de la clause, <ic> propose au gestionnaire de contrats de prendre en charge la négociation car la propriété négociée (`maxUsers`) est implémentée au niveau de son sous-composant <ig> (*étape 1*). Le contexte de la négociation atomique est alors transmis au gestionnaire de contrats de <ic> qui prend en charge la négociation, et consulte <ig> pour des efforts (*étape 2*). Par la suite, comme <ig> ne peut pas proposer de modifications à son niveau mais qu'il possède des éléments pour orienter la propagation de la négociation en exploitant la formule compositionnelle <ig>.maxUsers := <umgr>.maxUsers, il propose alors à son tour de propager la négociation. Cette formule compositionnelle traduit le fait que la propriété `maxUsers` implémentée au niveau de <ig> est réalisée au niveau de son sous-composant <umgr> ; c'est donc à ce dernier qu'il faut s'adresser si l'on souhaite exploiter jusqu'au bout la propagation de la négociation, et modifier effectivement la propriété `maxUsers` spécifiée dans la clause. Ainsi, la négociation est transmise au gestionnaire de contrats de <ig> (*étape 3*), qui exploite la formule compositionnelle pour s'adresser à <umgr> et obtenir de ce-dernier des propositions d'efforts visant à revalider la clause négociée (*étape 4*).

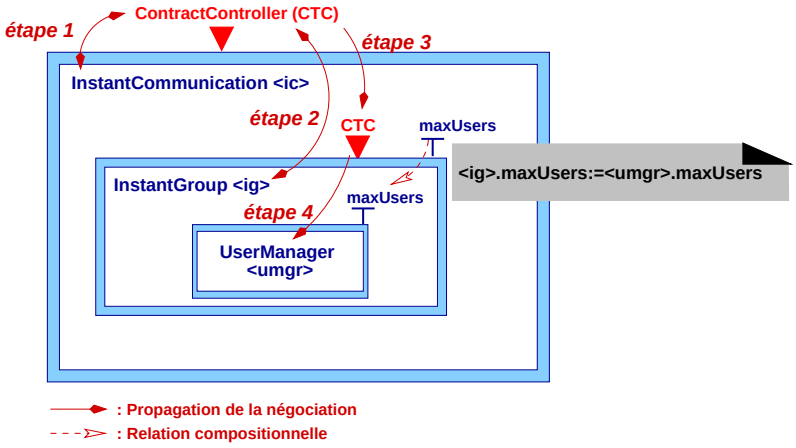


FIGURE 6.15 – *Schéma de propagation pour la propriété maxUsers*

Par la suite, compte tenu de ses propres capacités de négociation, `<umgr>` propose des modifications sur des valeurs de l'attribut `maxUsers`, qu'il supporte. Ainsi, avec sa liste d'alternative $\mathcal{A}_{inv,<umgr>} := \{\text{maxUsers} \leftarrow 150, \text{maxUsers} \leftarrow 250, \text{maxUsers} \leftarrow 300, \text{STOP}\}$, `<umgr>` propose alors successivement les modifications précédentes pour son attribut `maxUsers`. Pour chaque alternative proposée, le gestionnaire de contrats de `<ig>` effectue la modification et révérifie la clause pour la revalider. Dès qu'une alternative suffit à revalider la clause. Ici, comme le paramètre `max-users` de la clause vaut 250, la première modification suffisante est maxUsers $\leftarrow$ 250. La négociation atomique s'arrête alors, et comme l'attribut `maxUsers` est exactement la même sur `<umgr>` et sur `<ig>`, la clause initiale qui porte sur l'attribut `<ig>` est revalidée et se termine avec succès. Le contexte de négociation est alors retransmis au niveau de l'initiateur initial de la négociation pour que celui-ci finalise la négociation. Ainsi, la négociation qui est, au départ, déclenchée au niveau du composant `<ic>` est propagée dans l'intérieur de `<ig>` en exploitant le lien de réalisation, puis au niveau de `<umgr>` en exploitant la formule compositionnelle qui lie l'attribut `maxUsers` de `<ig>` à celui de `<umgr>`. `<umgr>` est alors consulté et ses capacités de négociation permettent de revalider la clause violée au niveau de `<ic>`.

Pour cette clause, la politique par concession permet, dans le cas le plus favorable, au composant bénéficiaire `<ic>` de relâcher la contrainte de la clause en agissant sur le terme paramétré de la clause (`max-users`). La politique par effort, quant à elle, permet au composant garant `<ig>`, après propagation et consultation du composant `<umgr>`, de re-satisfaire la clause par une action d'effort qui modifie l'attribut `maxUsers`.

6.6 Discussion

6.6.1 Retours sur les choix de conception

Différents choix de conception ont été effectués lors de la définition du modèle de négociation. Nous revenons sur les différents éléments du modèle, et introduisons divers traitements possibles en dehors des processus de négociation.

Négociation atomique. La négociation est dite atomique puisqu'elle se rapporte à l'élément atomique du contrat, la clause. Les parties en présence dans une négociation sont potentiellement ouvertes, mais

les politiques de négociation proposées s'appuient directement sur les informations de responsabilité de chaque clause. Pour chaque catégorie de responsables consultés, une politique est définie. La négociation atomique fait intervenir des entités qui sont liés par au plus un niveau de hiérarchie. Sa portée spatiale se limite donc aux composants impliqués visibles au niveau de hiérarchie donné. De plus, les négociations atomiques sont isolées les unes des autres et menées de façon indépendante, en accord avec les éléments de base qui définissent chacune d'entre elles.

Protocole de négociation. Lors de la négociation, les parties sont autonomes et leurs capacités sont privées. Pour répondre au besoin d'interactions entre ces parties, celles-ci sont d'une part réparties entre les composants responsables qui n'ont pas la vision des contrats et qui ne peuvent pas rétablir leur validité et, d'autre part, le gestionnaire de contrats qui dispose des connaissances pour contrôler la négociation. Le protocole de négociation est ainsi de cardinalité 1-N, dans lequel une partie pilote la négociation, et reprend les bases du *Contract-Net Protocol* (CNP). Il rend possible les interactions de toutes ces parties dans un contexte distribué tout en préservant leur autonomie. Il est ainsi est bien adapté à notre cadre, car comme nos parties ont beaucoup d'autonomie dans leurs propositions, le processus de négociation a besoin d'être contrôlé. L'utilisation du CNP, qui définit une partie spécifique dotée du rôle d'initiateur, permet ainsi d'apporter un minimum de contrôle au processus de négociation (vision hiérarchique, capacités de visibilité et de contrôle), tout en conservant l'encapsulation des composants au niveau concerné.

Parties négociantes. Du point de vue des parties impliquées, la politique par concession est une négociation de cardinalité 1-N qui met en jeu un initiateur et les N bénéficiaires principal et secondaire, alors que la politique par effort est de cardinalité 1-1 (au niveau de hiérarchie concerné par la violation) qui fait intervenir l'initiateur et l'unique garant. Au niveau de hiérarchie concerné par la violation, ces cardinalités découlent directement des responsabilités attribués par le modèle de contrats aux composants. Dans les cas où la négociation vient à se propager, une nouvelle négociation atomique se forme et la cardinalité des parties négociantes est déterminée par les composants qui interviennent dans les informations compositionnelles ; celle-ci peut donc varier à chaque niveau de propagation.

Politiques de négociation. Les politiques de négociation proposées ont deux rôles. D'une part, elles conduisent et contrôlent le processus de négociation car il est très ouvert. D'autre part, elles guident l'adaptation pour effectuer des modifications plus fines par rapport à des adaptations plus *classiques* (code d'adaptation arbitraire, consultation des parties par diffusion, etc.), en offrant la possibilité de faire des adaptations sur les contrats ou les composants avec ou sans propagation dans les hiérarchies. Ceci permet des paramétrages divers et différentes formes de déroulement de la négociation.

Dans la politique par concession, les responsables consultés sont les bénéficiaires. Ils bénéficient du fait qu'une clause est satisfaite et n'ont que la visibilité de celle-ci. Par conséquent, les bénéficiaires ne voient que la clause négociée et les propositions qu'ils sont susceptibles d'effectuer au cours de la négociation sont en rapport avec cette visibilité, et ne peuvent porter que sur des modifications de la clause négociée en entier ou sur certains de ses termes. La politique par effort aborde les choses différemment. Elle consulte le garant qui a une responsabilité liée à son assemblage ou son implémentation. Ici, le garant a une responsabilité en "valeur" sur la clause négociée. Lui ou ses sous-composants réalisent la propriété et les propositions qu'ils sont amenés à faire au cours d'une négociation peuvent alors consister en des reconfigurations (actions d'adaptations au niveau du garant d'assemblage, reconfiguration d'attributs des composants qui portent ou décomposent la propriété négociée). Les actions réalisables pour chaque politique sont ainsi très clairement déterminées à partir des responsabilités respectives.

Pour ce qui concerne l'application combinée des politiques par concession et par effort, les effets des deux politiques sont *a priori* non symétriques car elles n'impactent pas les mêmes éléments (modi-

fications des clauses vs. reconfiguration de composants). De plus, les actions réalisables dans chacune des deux politiques sont aussi individuellement très ouvertes, et rendent ainsi très difficile d'assurer la symétrie entre les actions réalisées par les bénéficiaires et celles du garant. De plus, le retour du système à des conditions initiales n'est pas prévu de façon explicite par le modèle de négociation. Pour permettre cela, il faudrait alors conserver une "memoire" des conditions initiales voire des modifications effectuées par les négociations successives pour une clause donnée d'un contrat, et aussi définir des mécanismes dédiés qui permettraient de les appliquer. Toutefois, de la même façon, comme les contrats ne conservent qu'une partie locale des évolutions du système, rien ne garantit le fait que ces conditions initiales puissent s'appliquer alors que le système se trouve dans un tout autre état.

6.6.2 Traitements divers lors d'échec de négociation

Le processus de négociation complet a été illustré précédemment selon les deux politiques par concession et par effort. En cas d'impossibilité (clause non négociable) ou d'échec de négociation (négociation non suffisante), les traitements suivants peuvent alors être mis en œuvre. Ces traitements sont surtout donnés ici en tant que mécanismes de rattrapage possibles dans et hors processus de négociation, pour aborder les cas limites du processus de négociation. Aucune évaluation qualitative de l'adéquation ou de l'efficacité de ces techniques n'est donnée ici. Une organisation, voire un pilotage avancé de ces techniques exigerait, de manière générale, plus de recul dans les mécanismes de traitements d'erreurs applicables, et une expertise développée pour les appliquer, pour un système particulier, aux erreurs détectées. Ainsi, il est possible d'effectuer les traitements suivants :

- à la fin d'une négociation atomique donnée, il est possible de redémarrer une nouvelle négociation atomique avec une politique de négociation différente. Un nouveau processus de négociation repart alors avec cette fois-ci des parties différentesa.
- lorsque la consultation des composants par la négociation atomique ne suffit pas à rétablir une clause valide, l'exécution de codes d'adaptation plus ouverts, et à des portées plus étendues peuvent être une solution. De tels codes seraient attachés au niveau du gestionnaire de contrats en charge du contrat violé et peuvent consister en des actions d'adaptations plus lourdes, comme des reconfigurations architecturales de composants (remplacements, ajouts...). Dans le cas de la clause du contrat de composition externe de notre exemple précédent qui porte sur une estimation de l'usage estimée de CPU du service de diffusion vidéo, le composant `VideoService` pourrait alors être remplacé par un autre composant présentant les même fonctionnalités mais conçu pour avoir une utilisation en CPU moindre, et ce remplacement serait effectué en utilisant les primitives de reconfigurations de la plate-forme *Fractal* (arrêt du composant, déconnexion des interfaces, ajout du nouveau composant, reconnexion des interfaces et redémarrage). Par la suite, comme le système *ConFract* suit les reconfigurations dynamiques des composants (cf. section B.4), les contrats sont mis à jour et le cycle de vie normal de l'application et du système de contrats repart au démarrage du composant.
- enfin, à tout moment, tout échec de négociation peut être signalé à des administrateurs du système pour qu'ils enisagent des traitements d'erreurs appropriés.

6.6.3 Contractualisation du système de négociation

Comme le modèle de négociation repose sur un modèle de contrats relativement général, et qu'il définit très précisément ses éléments de base, il serait possible de définir des contrats plus simples au niveau du système de négociation lui-même, ainsi des formes très affaiblies de négociation sur ces derniers (voir Fig. 6.16). De tels contrats peuvent, par exemple, permettre de spécifier *i)* des propriétés structurelles liées à la construction des négociations atomiques (nombre de participants à consulter, types des

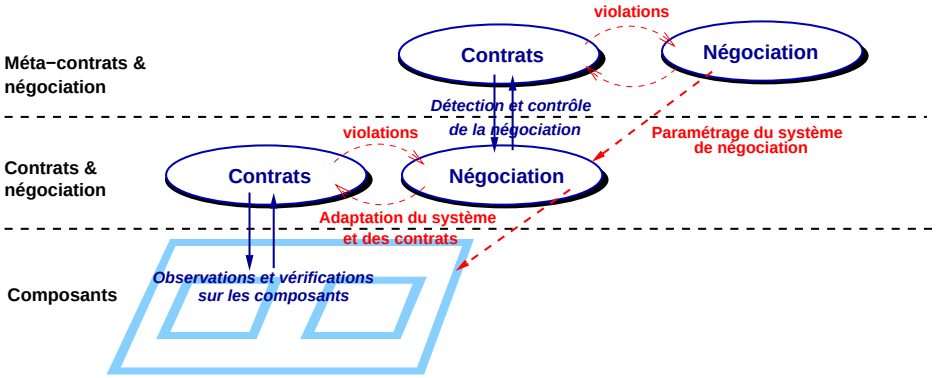


FIGURE *6.16 – Vision schématique des niveaux de contrats et négociation*

interfaces des parties négociantes...), *ii)* des enchaînements d'opérations à l'exécution des négociations atomiques (phase d'initialisation suivie d'une phase d'exécution, trois étapes d'interactions entre l'initiateur de la négociation et les participants...) et aussi *iii)* des propriétés qui permettent de paramétrer et contrôler les négociations atomiques. Cette dernière catégorie est la plus intéressante car elle peut notamment servir à spécifier des contraintes relatives à la terminaison des processus de négociation, telles que :

- un seuil maximal des durées des négociations atomiques ;
- un seuil maximal sur le nombre de niveaux de hiérarchies des négociations propagées ;
- un seuil maximal de renégociations d'une même négociation atomique donnée soit sur sur toute l'exécution du système, soit sur un intervalle de temps donné. Cela permet d'éviter que le système passe son temps à renégocier des clauses données, et de détecter des clauses sensibles qui sont fréquemment violées dans un intervalle de temps donné ;
- un seuil maximal de renégociations en cascade ou de cycle entre plusieurs clauses. En plus de la borne sur la durée de chaque négociation atomique individuelle, cela permet de confiner des renégociations entre clauses différentes mais *a postériori* dépendantes.

Comme la définition des contrats sur le système de négociation augmente la complexité en terme de niveaux d'abstraction introduits, les méta-contrats définis devront être relativement simples, et porteront typiquement sur des propriétés telles que celles précédemment présentées. Par ailleurs, le fait que le modèle de négociation proposé soit relativement simple et que les processus de négociation soient fortement orientés permet déjà, dans une certaine mesure, de gérer la complexité des processus de négociation par construction, mais cela contribue aussi à faciliter la définition de tels méta-contrats (entités bien identifiées, interactions simples...). De plus, en même temps que les contraintes sur le système de négociation vont être plus simples, les capacités de négociation vont être encore plus réduites et affaiblies qu'au niveau de base, et consisteront surtout à des modifications de seuils et des retraits de contrats. Enfin, il est à noter que comme le modèle de négociation proposé s'appuie par hypothèse sur des applications architecturées sous la forme de composants, une représentation, basée composants, du système de négociation permettrait de définir plus naturellement les méta-contrats, et contribuerait ainsi à manipuler contrats-et-négociation de façon uniforme, sur ces deux niveaux. Des détails sur la mise en œuvre basée composants du modèle de négociation sont donnés dans le chapitre 8.

7

Patrons d'intégration de propriétés extrafonctionnelles

7.1 Cadre d'étude

Comme nous l'avons souligné lors de l'analyse de l'état de l'art (cf. chapitre 5), actuellement, le spectre des travaux qui portent sur les aspects extrafonctionnels est très large et peut se classifier globalement entre ceux qui abordent les aspects extrafonctionnels de très haut-niveau et offrent alors peu de possibilités de gestion opérationnelles [Bertoa et Vallecillo 2002], et ceux qui mettent directement en œuvre des techniques de gestion ciblées souvent dédiés à certaines propriétés et sans réellement les expliciter [Stafford et McGregor 2002; Hamlet *et al.* 2001; Hissam *et al.* 2003; Wallnau *et al.* 2001]. Cette dichotomie met ainsi en évidence le manque de propositions à mi-chemin de ces deux classes de travaux et qui permettraient à la fois de décrire plus précisément les propriétés extrafonctionnelles des systèmes à composants (modélisation, spécification, etc.), et de les gérer de façon explicite lors des phases d'assemblages et d'exécution (réalisation dans le système en exécution, adaptation, etc.).

Dans ce cadre, et comme nous voulons capturer et gérer plus finement les aspects extrafonctionnels par des techniques de contrats et d'auto-adaptation, nous proposons dans ce chapitre une modélisation ainsi que des patrons d'intégration des propriétés extrafonctionnelles par rapport aux composants. Ces patrons permettent d'expliciter et d'uniformiser la réalisation d'un bon nombre de propriétés extrafonctionnelles au niveau des composants eux-mêmes, puisque ce sont ces derniers qui structurent et décomposent les systèmes à composants. Toutefois, compte tenu de la forte variabilité des aspects extrafonctionnels (cf. chapitre 3), notre étude ne va pas couvrir de façon exhaustive les diverses propriétés extrafonctionnelles. Au contraire, nous nous intéressons surtout aux propriétés extrafonctionnelles définies de façon suffisamment précise, et qui décrivent des propriétés observables et mesurables des composants. En particulier, elle inclut celles qui varient à l'exécution et sur lesquelles il est pertinent de faire des observations, vérifications et adaptations avec les techniques de contractualisation et d'auto-adaptation présentées dans notre démarche (cf. chapitre 1). En revanche, les aspects extrafonctionnels de haut-niveau ou qui concernent des propriétés d'évolution des systèmes [Barbacci *et al.* 1995] (sécurité, fiabilité, robustesse, maintenabilité,...) sont mis de coté, car elles ne concernent pas directement notre problématique et du fait de leur généralité sont plus difficiles à décrire, observer et mesurer. De même, les aspects temporels, qui nécessitent une modélisation à la fois plus fine et liée à des évènements fonctionnels, ne sont pas étudiés ici. Par ailleurs, notre point de départ étant les systèmes construits par

assemblage de composants logiciels, cette étude se concentre aussi sur les propriétés extrafonctionnelles orthogonaux aux aspects fonctionnels et qui peuvent être liés individuellement aux composants. Ainsi, nous souhaitons en plus rendre le plus explicite possible un large éventail de propriétés extrafonctionnelles qui découlent naturellement des composants pris individuellement.

Dans ce cadre d'étude, notre démarche ici consiste ainsi à analyser certaines caractéristiques des propriétés extrafonctionnelles, afin d'établir une classification qui prenne en compte à la fois les relations des propriétés avec les composants ainsi que leur cycle de vie. À ces diverses classes de propriétés extrafonctionnelles, nous associons plusieurs patrons architecturaux qui permettent d'intégrer ces propriétés au niveau des éléments d'architecture usuels des composants (composant, interface, attribut). Par la suite, une fois que les réalisations des propriétés extrafonctionnelles sont clairement modélisées vis-à-vis des composants, nous nous intéressons aussi à définir des formes de raisonnement compositionnel permettant de décrire la réalisation des propriétés d'un assemblage de composants compte tenu de propriétés dans sa décomposition.

7.2 Classification

7.2.1 Analyse

Notre étude des propriétés extrafonctionnelles se focalise sur les deux dimensions suivantes. Pour pouvoir analyser les propriétés extrafonctionnelles des systèmes à composants, nous étudions les relations qu'elle sont vis-à-vis des composants eux-mêmes (« *quelles sont celles qui sont directement liées à des composants individuels ?* », « *quelles informations des composants expriment-elles ?* »). De plus, pour pouvoir être observées et évaluées finement, les propriétés extrafonctionnelles doivent être mesurées. Pour ce faire, nous étudions alors aussi le cycle de vie des propriétés extrafonctionnelles en fonction de celui des composants (« *à quels moments ces propriétés sont-elles définies ?* », « *à quels moments peuvent/doivent-elles être observées ?* », « *quelle est la validité de ces observations ?* »). L'analyse des propriétés extrafonctionnelles, selon la première dimension servira à intégrer et modéliser très précisément une propriété extrafonctionnelle donnée à des composants individuels, alors que l'étude de la seconde dimension apporte des informations sur le cycle de vie des propriétés nécessaires à leurs observations (moments de définition, moments d'observation, durée de validité, etc.).

Ainsi selon ces deux dimensions, une première catégorie de propriétés extrafonctionnelles est formée de celles qui représentent des informations clés liées à la *nature des composants*. Par exemple, elles décrivent une empreinte mémoire, une version de compatibilité d'un codec vidéo ou encore une capacité maximum d'un disque ou d'une mémoire. Ces propriétés peuvent être assimilées aux caractéristiques techniques des composants électroniques ou mécaniques. Elles capturent et représentent des caractéristiques importantes liées à la conception ou au développement des composants et ont définitivement un impact sur leurs usages et leurs exécutions futurs. Ainsi, comme ces propriétés découlent de certains choix de fonctionnalités ou d'implémentation des composants, elles sont définies au moment de la conception. De plus, leur observation devient pertinente surtout pendant la phase d'assemblage lorsqu'il s'agit de prendre en compte leurs caractéristiques intrinsèques lors des assemblages pour vérifier l'adéquation aux besoins ou pour effectuer des tests de compatibilité (contrôle d'admission). Par ailleurs, comme elles concernent des caractéristiques de la nature des composants, les valeurs mesurées restent constantes et valides entre deux développements successifs (constantes à la configuration et exécution).

D'autres propriétés des composants représentent leurs *paramètres de configuration*. Ces propriétés décrivent par exemple la taille d'un composant de *buffer*, le port d'écoute d'un serveur web, ou encore la taille maximum d'un *pool* de ressources. Elles influencent les services requis et fournis par les composants et sont souvent (re-)paramétrées pour définir des modes de fonctionnement différents des composants. Ces paramètres de configuration peuvent être définis lors de la phase de développement par

certaines valeurs par défaut, mais très souvent ils sont observés et modifiés lors de la phase d'assemblage et de configuration pour définir des modes de fonctionnement particuliers des composants ou les adapter afin de prendre en compte un environnement d'exécution donné (contexte des autres composants, infrastructure d'exécution, etc.). Une fois définies, ces propriétés restent généralement constantes entre deux phases de reconfiguration. De plus, comparées aux propriétés de nature qui restent constantes, comme les paramètres de configuration des composants sont définis pour offrir différents fonctionnements aux composants, ils sont modifiés plus fréquemment lors des phases de (re-)configurations, et restent constants entre deux phases de configuration successives, y compris donc à l'exécution.

Dans une troisième catégorie, on trouve aussi des propriétés qui décrivent le *fonctionnement* des composants. Par exemple, elles décrivent le nombre de sessions concurrentes d'un serveur web, l'état courant d'un lecteur vidéo lors du traitement du flux des données, ou encore le nombre de paquets courants échangés entre un client et un serveur. Ces propriétés capturent des informations clés du fonctionnement des composants lors de leur exécution et peuvent donc être assimilées à des sondes qui collectent certaines informations fonctionnelles des composants. Comme elles sont liées au fonctionnement des composants, ces propriétés sont naturellement définies et observées lors de l'exécution. De plus, contrairement aux deux catégories précédentes, ces propriétés de fonctionnement se focalisent plus sur le comportement fonctionnel des composants.

Les deux dernières catégories sont formées par des propriétés liées à l'infrastructure d'exécution. La quatrième catégorie est représentée par les propriétés qui décrivent des ressources physiques telles que la mémoire, le processeur (CPU) ou encore d'autres caractéristiques relatives au support réseau telle que la bande-passante. Ces propriétés, usuellement regroupées sous la terminologie de *ressources*, représentent des propriétés liées à la plate-forme d'exécution et traduisent les besoins exogènes des applications. Elles déterminent clairement les qualités d'exécution des services en terme de ressources fournies par l'infrastructure aux applications, et doivent être souvent réifiées au niveau des applications pour être surveillées et mieux gérées. Comme les ressources conditionnent l'exécution des systèmes, elles ne sont définies entièrement qu'au moment du déploiement, une fois l'environnement d'exécution connu. Leur existence est constante pendant une exécution donnée et ce jusqu'à un prochain déploiement. Au delà de l'existence propre des ressources, ce sont davantage les propriétés relatives à leur capacité qui sont importantes. Les *propriétés de capacité* décrivent des aspects de consommation de ressources, comme les consommations en mémoire ou en batterie. Comme elles expriment les niveaux de ressources fournis aux applications, elles constituent alors la dimension des ressources la plus souvent considérée et sont importantes pour les applications sensibles aux ressources. Ces propriétés conditionnent ainsi les qualités d'exécution des systèmes, et représentent les éléments critiques de certains systèmes. Ainsi, elles déterminent les besoins en terme de niveaux de ressources requis des applications. Elles sont définies et observées lors de l'exécution des systèmes. De plus, elles peuvent évoluer de façon très variable, en étant plus ou moins fluctuante, et *a priori* sans modèle d'évolution prévisible.

7.2.2 Synthèse

Sous les hypothèses de notre étude, nous avons identifié les catégories de propriétés extrafonctionnelles suivantes : *(i)* les propriétés de *nature* des composants, *(ii)* les propriétés de *configuration* des composants, *(iii)* les propriétés de *fonctionnement* des composants, *(iv)* les *ressources* physiques réifiées au niveau applicatif, *(v)* les propriétés de *capacité* de ces ressources. Ces propriétés sont établies en étudiant celles qui sont directement liées aux composants individuels, et qui expriment des propriétés relativement orthogonales à leurs aspects fonctionnels. De plus, pour pouvoir être observées et évaluées plus finement, le cycle de vie de chacune d'entre elles est aussi précisé et décrit notamment les moments auxquels elles sont définies, les moments auxquels les observations sont valides et pertinentes, et les phases de validité de leurs observations. Le tableau 7.1 qui suit synthétise les catégories de propriétés

	Exemples	**Définition**	**Observation**	**Evolution**	**Validité**
Attributs de nature	version de compatibilité, empreinte mémoire	conception	configuration	constant à la configuration et exécution	toute une configuration et une exécution
Propriétés de configuration	taille d'un *buffer*, capacité maximum d'un *pool* de ressources, port d'écoute d'un serveur	conception, configuration, reconfiguration	configuration, reconfiguration	constant à l'exécution	toute une exécution
Paramètres de fonctionnement	Nombre de sessions concurrentes, nombre de paquets échangés	exécution	exécution	modifié à l'exécution	entre deux modifications non prévisibles *a priori*
Ressources	batterie, mémoire, processeur	déploiement	déploiement, exécution	existence constante à l'exécution	entre deux déploiements
Propriétés de capacité	consommation de ressources (niveau de batterie, consommation mémoire, etc.)	exécution	exécution	variable à l'exécution	aucune *a priori*

TABLE *7.1 – Synthèse des catégories de propriétés extrafonctionnelles et de leur cycle de vie.*

extrafonctionnelles proposées.

7.3 Modélisation

Afin d'expliciter la réalisation des catégories de propriétés extrafonctionnelles présentées précédemment, un patron d'intégration est défini pour chacune d'entre elles. Les patrons proposés sont décrits de façon abstraite en utilisant les diagrammes de composants d'*UML* 2 afin de conserver une certaine indépendance vis-à-vis des technologies à composants sous-jacentes.

Les trois premières catégories de propriétés extrafonctionnelles (attributs de nature, paramètres de configuration et paramètres de fonctionnement) correspondent directement aux concepts d'attributs de composants. Plus précisément, comme les *attributs de nature* ne sont pas modifiables, ils sont modélisés par des attributs de composants accessibles en lecture seule. Les valeurs de ces attributs sont alors récupérés par le biais des opérations d'accès définis dans une interface fournie correspondante. Les *paramètres de configuration* peuvent avoir des valeurs par défaut et être re-paramétrisés. Ils sont donc aussi naturellement modélisés par des attributs accessibles à la fois en lecture et en écriture, et ils sont accédés par les opérations de lecture et d'écriture définis dans une interface fournie correspondante. Les *paramètres de fonctionnement* quant à eux fournissent des informations sur certains aspects liés aux fonctionnement des composants. Ils ne sont donc pas modifiables et sont modélisés, de la même façon que les attributs de nature, par des attributs de composants accessibles en lecture seule, par le biais des opérations de lecture. Il est à noter que du seul point de vue de leur modélisation, les patrons d'intégration proposés pour les trois catégories précédentes sont structurellement très similaires, et consistent en des attributs de composants accessibles en lecture seule ou, en lecture et écriture. En revanche, les sémantiques d'utilisation de chaque patron sont radicalement différentes. Elles définissent chacune des moments particuliers auxquels elles sont définies et observées (cf. tableau de synthèse 7.1).

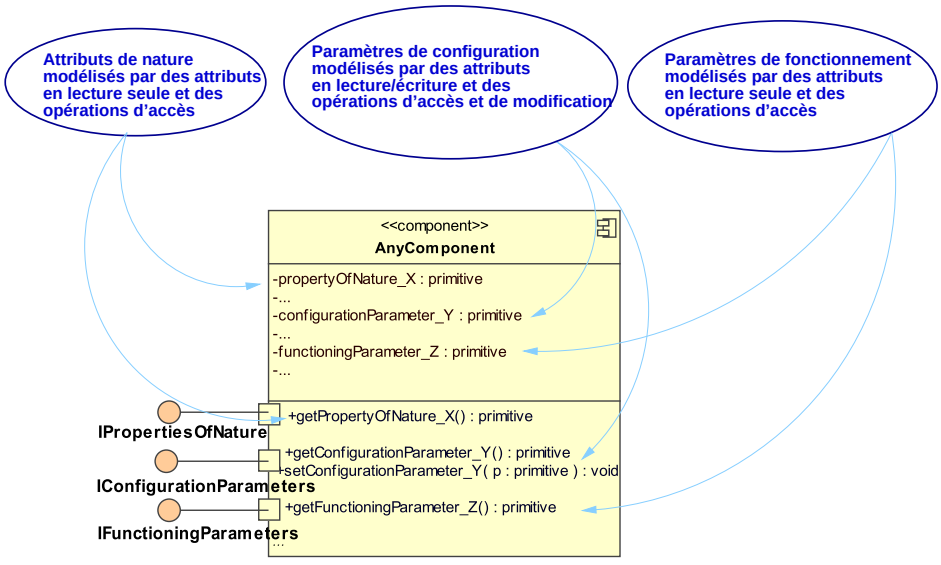


FIGURE 7.1 – *Patrons pour les attributs de nature, les paramètres de configuration et les paramètres de fonctionnement*

Les *ressources physiques* représentent des éléments de l'infrastructure sous-jacente. Afin de conserver cette structure à la fois au niveau de l'infrastructure et de l'application, celles-ci sont alors réifiées au niveau applicatif par des composants à part entière. Cela permet ainsi d'utiliser ces composants de ressources de la même façon que les composants métiers d'une façon uniforme, et de les exploiter pour

récupérer diverses informations sur les ressources physiques qu'ils réifient. En particulier, comme les *propriétés de capacité* représentent de telles informations et que les ressources physiques sont représentées par des composants à part entière, les propriétés de capacité sont modélisées par des opérations des composants de ressources et accédés au travers d'interfaces fonctionnelles des composants de ressources correspondants.

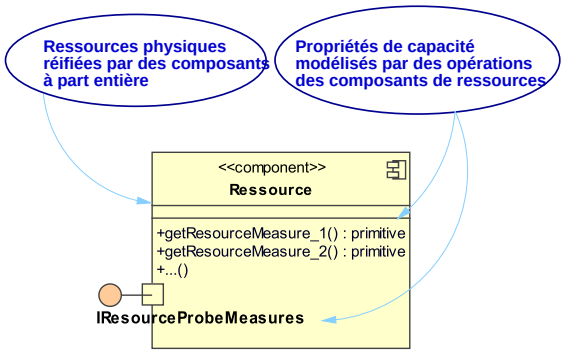


FIGURE 7.2 – *Patrons pour les ressources et les propriétés de capacité*

Il est à noter que comme les catégories présentées décrivent des propriétés distinctes, les patrons d'intégration associés peuvent être appliqués simultanément lorsque des composants exhibent des propriétés qui appartiennent à plusieurs de ces catégories. Un composant métier qui possède à la fois des attributs de nature, des paramètres de configuration et des paramètres de fonctionnement appliquerait ainsi les patrons d'intégration de ces trois catégories. C'est aussi le cas des composants de ressources qui réifient les ressources physiques puisqu'ils représentent des composants à part entière. Un composant de ressources qui expose des propriétés de nature ou de fonctionnement, appliquerait les patrons d'intégration correspondants. Par ailleurs, comme les patrons d'intégration proposés modélisent des propriétés extrafonctionnelles qui sont liées individuellement aux composants, ces patrons peuvent être appliqués sur des composants primitifs ou composites. Enfin, pour ce qui concerne les aspects de mesure, les patrons décrits proposent surtout des moyens standards pour expliciter la réalisation des propriétés extrafonctionnelles au niveau des composants et ne définissent pas de mécanismes permettant de les mesurer. Toutefois, des mécanismes ou systèmes de mesure et surveillance pourraient justement exploiter ces patrons dans la mesure où ces derniers explicitent clairement les propriétés et précisent les moments d'observation et de mesure associés.

7.4 Raisonnement compositionnel

7.4.1 Description compositionnelle

Après avoir classifié un sous-ensemble des propriétés extrafonctionnelles et proposé des patrons qui permettent de les intégrer aux composants, nous étudions maintenant dans quelle mesure il est possible de décrire et *raisonner* compositionnellement sur des propriétés extrafonctionnelles par rapport aux *compositions de composants*. En effet, les patrons que nous avons proposé explicitent la réalisation des propriétés extrafonctionnelles au niveau des composants mais ne décrivent pas les relations potentielles qui peuvent exister entre les propriétés des composants. Comme les composants peuvent être assemblés

à différents niveaux de hiérarchie, ce point est particulièrement important et il notamment pertinent de fournir des moyens qui permettent de raisonner sur des propriétés d'un assemblage résultant compte tenu des composants dans sa composition.

Dans notre étude, les propriétés extrafonctionnelles sont liées individuellement aux composants, et comme les patrons d'intégration servent déjà à expliciter leur réalisation par rapport aux composants, primitif ou composite, nous prolongeons notre étude dans le but de décrire et fournir des moyens qui permettent de raisonner sur de telles propriétés compositionnelles. En plus d'expliciter clairement la réalisation des propriétés extrafonctionnelles individuellement par rapport aux composants, leur description compositionnelle permet alors de comprendre structurellement comment celles-ci se décomposent dans des hiérarchies de composants, et rend possible la déduction des propriétés des assemblages compte tenu de leurs compositions.

A l'exception des propriétés de ressources qui ne décrivent pas de propriétés quantifiables, les quatre autres catégories de notre classification sont modélisées par des patrons qui sont liées exclusivement aux composants individuels. Ces propriétés sont donc compositionnelles par nature et il est possible de fournir une description ainsi qu'un support compositionnel pour ces propriétés. Pour cela, nous définissons comme étant *compositionnelle* une propriété qui fournit les informations suivantes : *C1)* les propriétés qui contribuent à sa décomposition, *C2)* pour chacune de ces propriétés, les composants qui la réalisent, *C3)* une fonction compositionnelle qui permet de calculer la valeur de la propriété compositionnelle compte tenu des propriétés qui la décomposent.

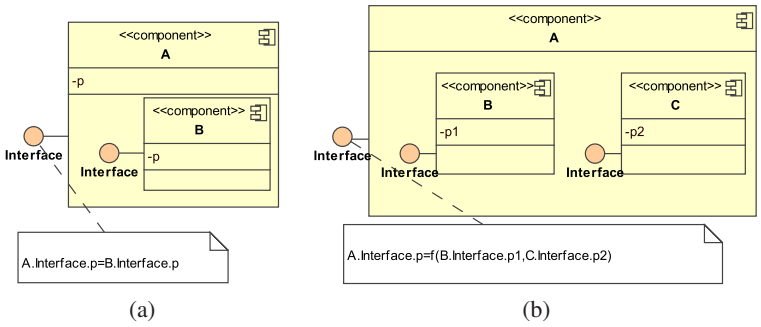


FIGURE *7.3 – Exemples de relations compositionnelles*

La figure 7.3 donne deux exemples de propriétés compositionnelles simples. La figure 7.3a décrit par la formule compositionnelle la relation entre la propriété `p` du composant `A` et la même propriété `p` de son sous-composant `B`. Ainsi, la propriété `p` au niveau de `A` se décompose selon la même propriété `p`. Cette dernière propriété est réalisée au niveau de `B` et les deux propriétés sont liées par une fonction identité. Dans la figure 7.3b, la relation compositionnelle permet cette fois-ci de décrire la propriété `p` de `A` par rapport aux propriétés `p1` et `p2` réalisées respectivement sur `B` and `C`. Les propriétés qui décomposent `p` sont `p1` et `p2` ; elles sont réalisées par les composants `B` et `C` respectivement, et liées par la fonction compositionnelle `f`, qui représente par exemple une fonction arithmétique simple (somme, min, etc.).

Dans le cas général, il est très difficile, voire impossible, de décrire formellement de telles *informations compositionnelles*, en particulier, le verrou majeur vient de la difficulté d'exprimer explicitement la fonction compositionnelle (*C3*). Dans notre étude, l'identification de ces informations est rendue plus simple car d'une part nous nous sommes limités aux propriétés simples et directement liées à des composants individuels, et d'autre part parce que les propriétés compositionnelles considérées se décrivent exclusivement selon d'autres propriétés de composants. Il y a conservation des propriétés sans influence de l'environnement, de l'architecture ou encore des usages qui entrent en jeu et doivent être pris en compte. Toutefois, en dépit de ces simplifications, et compte tenu de la nature même des propriétés ex-

trafonctionnelles, de nombreuses dépendances avec d'autres propriétés extrafonctionnelles peuvent toujours exister au niveau de leur décomposition ou de leur mesure notamment, et elles ne peuvent alors pas être exprimées et modélisées par ces simples fonctions compositionnelles. Les fonctions compositionnelles doivent alors être définies individuellement pour chaque propriété considérée et il est important de trouver un équilibre entre des relations compositionnelles prouvées mais difficiles à établir et s'appliquant dans des contextes bien définis, et des fonctions compositionnelles plus simples mais moins précises. Dans notre cas, les relations compositionnelles peuvent être définies pratiquement avec du code arbitraire, et nous faisons le choix de décrire ces informations compositionnelles par des méta-données intégrées par annotation des opérations des interfaces correspondantes ou par des DSL.

7.4.2 Support compositionnel

Comme les composants n'intègrent *a priori* aucun support permettant de gérer leurs propriétés compositionnelles, pour pouvoir raisonner sur celles-ci à l'exécution, nous définissons spécialement de nouvelles entités (au niveau méta) indépendamment des plates-formes de composants.

Entités

Chaque propriété compositionnelle est réifiée par un méta-objet dédié (*CompositionalProperty*) afin de donner accès à ses informations compositionnelles. Ces objets sont gérés par un gestionnaire de propriétés compositionnelles (*CompositionalPropertyManager*) défini pour chaque composant dans le but de gérer l'ensemble des propriétés compositionnelles de ce dernier. À l'instanciation d'un composant qui possède des propriétés compositionnelles, ce gestionnaire exploite les informations compositionnelles fournies pour instancier et enregistrer les méta-objets associés à chaque propriété compositionnelle. En phase de fonctionnement, il est ensuite utilisé pour retrouver le méta-objet d'une propriété sur laquelle on souhaite raisonner et obtenir des informations compositionnelles. Ainsi, chaque méta-objet est utilisé pour fournir les diverses informations compositionnelles sur la propriété compositionnelle qu'il réifie (propriétés contributrices, composants contributeurs, valeur, fonction compositionnelle, etc.). Ces informations compositionnelles doivent être préalablement fournies par exemple au moyen d'annotations des méthodes des interfaces concernées, ou de façon plus abstraite, en utilisant un langage dédié (DSL). Elles décrivent : *(i)* la liste des propriétés qui décomposent la propriété compositionnelle, *(ii)* pour chaque propriété la liste des composants et des éléments de réalisation (opération, interface), *(iii)* la fonction compositionnelle qui lie les propriétés contributrices à la propriété compositionnelle. La figure 7.4 fournit une vision schématique des entités définies.

Intégration avec les composants

Le gestionnaire des propriétés compositionnelles est l'élément central et est défini pour un composant (*Component*) qui possède des propriétés compositionnelles. Ce gestionnaire utilise des informations du contexte du *Component* pour instancier, par la *CompositionalPropertyFactory*, les méta-objets *CompositionalProperty*. Chaque *CompositionalProperty* se décompose alors en une ou plusieurs autres propriétés *QuantitativeNonFunctionalProperty* (compositionnelle ou non), chacune étant attachée à un composant *Component* et fournissant des éléments d'identification basiques sur sa déclaration (nom de l'interface, nom de l'opération, etc.). Tous ces éléments sont construits lors de l'initialisation des composants, et à l'exécution, le gestionnaire de chaque composant est ensuite utilisé, tel un registre par des clients du niveau de base ou du méta-niveau, pour récupérer les instances des méta-objets réifiant chaque propriété compositionnelle.

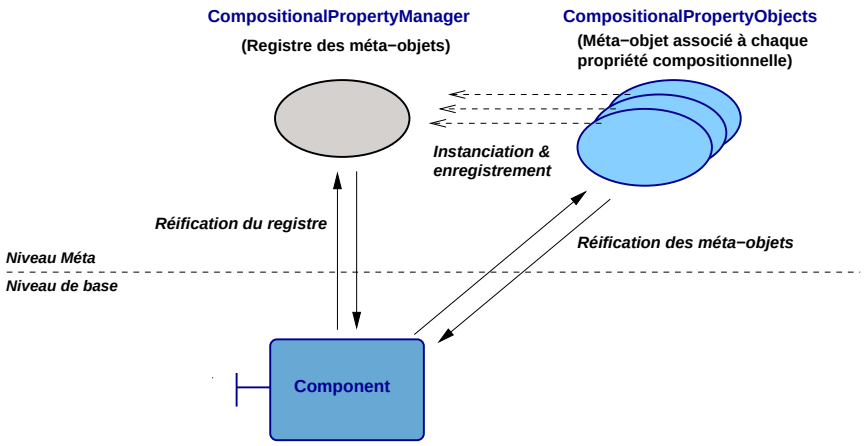


FIGURE 7.4 – *Architecture conceptuelle des entités définies*

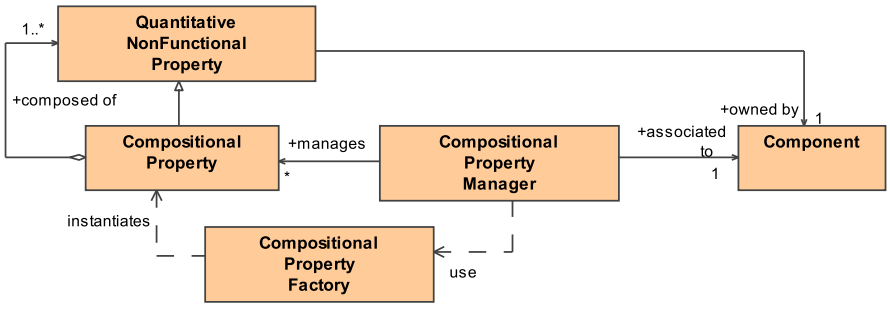


FIGURE 7.5 – *Support du raisonnement compositionnel*

Interactions

Les diagrammes de séquences qui suivent illustrent les interactions lors de l'initialisation et de l'exploitation du gestionnaire des propriétés compositionnelles et des méta-objets. La figure 7.6 décrit les opérations pour l'instanciation et l'enregistrement des propriétés compositionnelles.

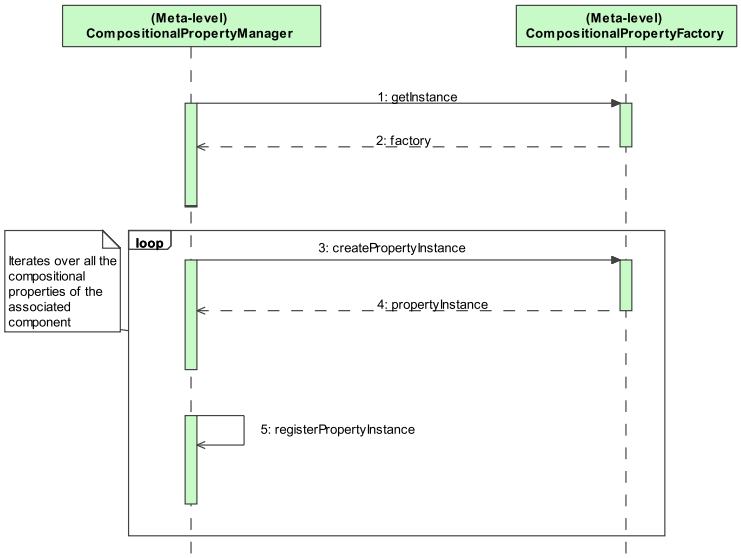


FIGURE 7.6 – *Instanciation des propriétés compositionnelles et initialisation du gestionnaire*

Une fois le *CompositionalPropertyManager* initialisé lors de l'instanciation des composants, il est utilisé, en phase d'exécution, par des clients à la fois du niveau de base ou méta (cf. Fig. 7.7). Ce gestionnaire peut être utilisé par un composant métier qui souhaite obtenir des informations basiques sur les propriétés compositionnelles (valeur de la propriété compositionnelle, etc.), ou par toute autre entité qui souhaite observer, mesurer ou raisonner sur la propriété compositionnelle (ses éléments de réalisations) ou sa décomposition structurelle (propriétés contributrices et leurs éléments de réalisation).

7.5 Projection dans la plate-forme Fractal

7.5.1 Patrons d'intégration

Le modèle *Fractal* propose une interface de contrôle spécifique qui modélise et permet de définir et de gérer les propriétés orthogonales primitives des composants [Bruneton *et al.* 2002]. Cette interface permet de donner accès aux attributs des composants sans qu'il ne soit nécessaire de les démarrer ou de connecter les autres interfaces fonctionnelles des composants. De plus, elle offre aussi différents modes d'accès – lecture et/ou écriture – et cela permet donc de respecter la différences entre les catégories de notre classification précédente dans laquelle les propriété de nature et les paramètres de fonctionnement ne sont modifiables, alors que les paramètres de configuration le sont davantage (voir section 7.2). Ainsi, comme les *propriétés de nature*, les *paramètres de configuration* et les *paramètres de fonctionnement* représentent des attributs de composants, leur modélisation se fait naturellement par des attributs *Fractal* et une interface de contrôle des attributs (voir Fig. 7.8a and 7.8b), avec leurs opérations de lecture/écriture associées.

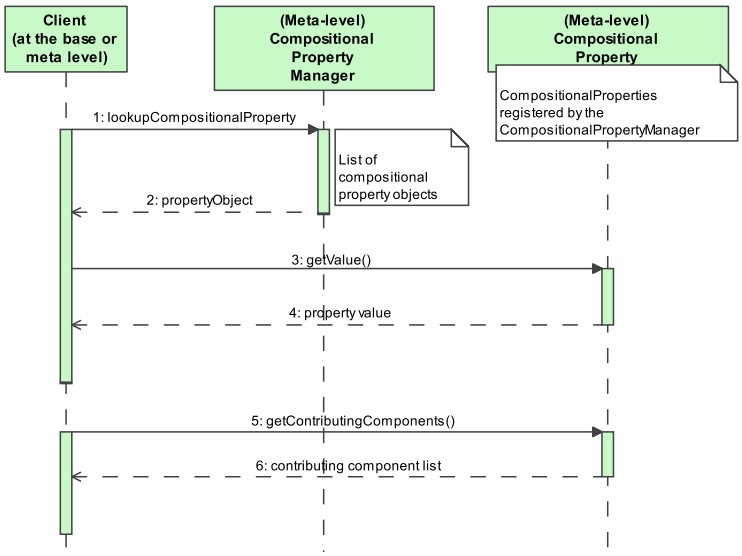


FIGURE 7.7 – *Interactions entre un client (base ou méta) avec les entités du meta-niveau*

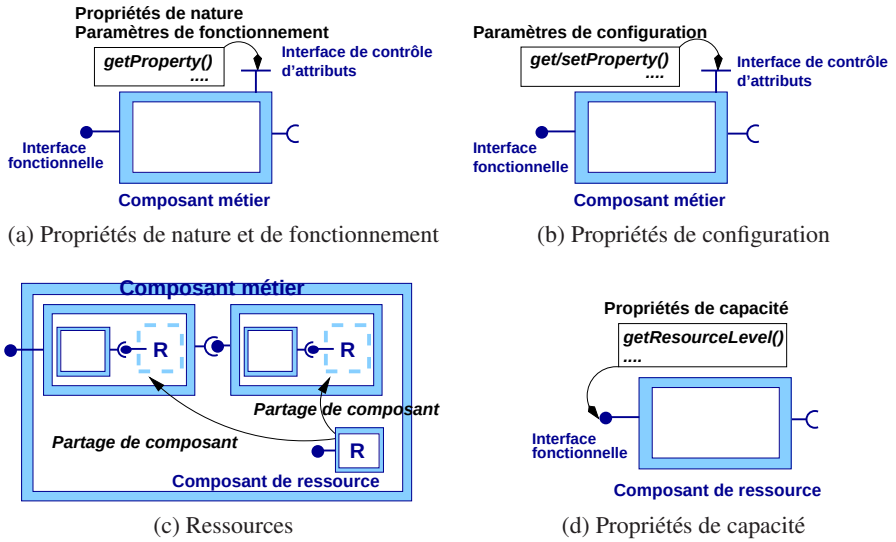
Comme les *ressources* sont modélisés par des composants, ils sont, dans la plate-forme *Fractal*, aussi réifiés sous la forme de composants à part-entière (voir Fig. 7.8c). Compte tenu de la diversité des ressources physiques, les composants de ressources qui les réifient ne fournissent pas *a priori* de fonctionnalités élaborée, si ce n'est de collecter diverses informations sur l'état des ressources physiques. En particulier, comme les *propriétés de capacité* des ressources constituent de telles informations basiques collectées par des sondes de ressources, lesquelles sont modélisées par des composants, ces propriétés sont modélisées par des interfaces fonctionnelles des composants de ressources associées au composant de ressource (voir Fig. 7.8d). La modélisation n'impose pas de contraintes spécifique sur la nature des données de sondes collectées. Celles-ci restent ouvertes, et doivent donc être déterminées par les développeurs. Par exemple, ces mesures peuvent être très primitives et collectées directement au niveau des ressources physiques, ou alors être plus complexes issues de calcul plus statistiques plus sophistiqués (interpolations, corrélations, etc.).

Enfin, il est à noter que comme des ressources physiques peuvent être utilisées par différents composants, le *partage* des composants [Bruneton *et al.* 2002] offert par la plate-forme *Fractal* est ici reprise pour refléter au niveau applicatif le partage de ressources physiques, mais aussi cela permet d'utiliser une même instance de composant de ressource, dans des composites distincts, potentiellement à différents niveaux de hiérarchie, tout en préservant l'encapsulation des composants.

7.5.2 Support compositionnel

Compte tenu de l'ouverture de la plate-forme *Fractal*, la gestion des propriétés compositionnelles des compositionnelles est aisément intégrée sous la forme d'une nouvelle fonctionnalité de contrôle et celle-ci est effectuée en suivant les principes d'intégration des contrôleurs définis dans *Julia*. Le gestionnaire de propriétés compositionnelles (*CompositionalPropertyManager*) est donc implémenté sous la forme d'une nouvelle interface de contrôle *Fractal*, intégrée dans la membrane de tout composant composite paramétré. Le fonctionnalités offertes par ce nouveau contrôleur peuvent être utilisées de façon usuelle en récupérant par introspection l'interface de contrôle à partir de son nom.

Ce contrôleur gère l'ensemble des propriétés compositionnelles du composite sur lequel il est défini,



(a) Propriétés de nature et de fonctionnement

(b) Propriétés de configuration

(c) Ressources

(d) Propriétés de capacité

FIGURE 7.8 – *Récapitulatif des patrons d'intégrations dans la plate-forme Fractal*

et maintient, pour cela, une table des différents objets réifiant les propriétés compositionnelles (*CompositionalProperty*). Chacun de ces objets donne notamment accès à 1) la liste de tous les objets *QuantitativeNonFunctionalProperty* décomposant la propriété compositionnelle, et à 2) la valeur de cette dernière. Les objets *QuantitativeNonFunctionalProperty* sont instanciés à partir des informations compositionnelles décrites par des annotations *Java* multi-valuées sur l'interface d'attributs, et fournissent chacun les diverses informations de la propriété contributrice (nom, composant, interface, méthode, valeur). En particulier, la valeur de la propriété compositionnelle (*CompositionalProperty*) est calculée en récupérant les valeurs des différentes propriétés contributrices *QuantitativeNonFunctionalProperty*, et en évaluant ces valeurs selon la fonction compositionnelle.

Il est à noter que comme une propriété compositionnelle d'un composite dépend des propriétés de ses sous-composants, le gestionnaire de propriétés compositionnelles doit être notifié des événements d'ajouts de sous-composants. Pour chaque propriété compositionnelle, l'instanciation effective des objets *CompositionalProperty*, *QuantitativeNonFunctionalProperty* s'opère seulement lorsque tous les sous-composants requis sont disponibles dans le composite concerné. Inversement, dès qu'un sous-composant qui intervient dans la décomposition d'une propriété compositionnelle est retiré, la description compositionnelle de la propriété n'a plus aucun sens et l'objet qui réifie la propriété compositionnelle est invalidé.

8

Mise en œuvre dans Fractal et ConFract

Le chapitre 6 a défini un modèle de négociation dynamique de contrats pour composants logiciels contractualisés qui s'appuie sur les principes généraux des modèles de composants *Fractal* et de contrats *ConFract*. Le système de négociation proposé offre comme principale fonctionnalité de rétablir la validité des contrats en faisant négocier les composants eux-mêmes. Il travaille sur les clauses, éléments les plus atomiques d'un contrat, et s'organise autour des négociations atomiques. Ces négociations atomiques regroupent les différents éléments de base nécessaires à toute négociation :

- Des *parties* bien définies : le contrôleur de contrats en tant qu'initiateur, les composants impliqués dans la clause avec leur responsabilité déterminée par le système *ConFract*, un éventuel participant externe pour injecter des paramètres de pilotage de plus haut niveau ;
- Des *objets de négociation* que sont les clauses des contrats autour desquels ces parties interagissent ;
- Des interactions entre parties organisées selon un *protocole* de négociation qui reprend les principes généraux du *Contract-Net Protocol* ;
- Différentes *politiques* de négociation qui peuvent exploiter des informations fines contenues dans les contrats comme les responsabilités des composants et qui fournissent alors différentes orientations aux négociations atomiques.

La suite de ce chapitre détaille la mise en œuvre du modèle de négociation dans les plates-formes *ConFract* et *Fractal*. Les besoins en termes d'intégration vis-à-vis des plates-formes et du cycle de vie des composants et des contrats sont rapidement décrits. Puis, l'architecture et le fonctionnement plus précis du système sont présentés.

8.1 Définition des besoins

L'intégration du système de négociation se fait sur une implémentation de la plate-forme *ConFract*, elle-même basé sur l'implémentation de référence *Julia* du modèle *Fractal*. De plus, comme le système de négociation étend le système de contrats construit sur les composants *Fractal*, les processus de négociation définis ont besoin de s'appuyer sur plusieurs éléments extraits des contextes des composants et des contrats, et ils doivent aussi s'intégrer dans le cycle de vie des composants et des contrats.

Interfaçage avec Fractal

Le modèle de négociation utilise un certain nombre de fonctionnalités du modèle de composants *Fractal* dans sa version 2 [Bruneton *et al.* 2002]. En particulier, les informations sur les composants (recherche des participants lors du processus de négociation, objets de la négociation) sont récupérées par introspection des composants. De plus, les capacités supplémentaires nécessaires à la négociation (raisonnement lors de la négociation, informations compositionnelles éventuellement exploitées) sont intégrées à l'aide de la notion de contrôleur dans la membrane des composants (cf. section 7.5, page 106). L'implémentation du modèle *Fractal* choisie pour l'intégration est celle de référence *Julia*. Dans sa version 2.5, *Julia* offre notamment la possibilité d'implémenter les contrôleurs sous la forme des composants *Fractal*. Ceci permettra de placer des composants *Fractal* relatifs à la négociation dans les membranes de composants, tout en étant aussi compatible avec une version objet du contrôleur de contrats *ConFract*.

Interfaçage avec ConFract

Vis-à-vis du système de contrats, le modèle de négociation s'intègre à la première version du système *ConFract*, et les processus de négociation reposent sur les hypothèses suivantes. La capacité de négociation est intégrée par extension du contrôleur de contrats. En effet, aucun nouveau *contrôleur de négociation* n'est intégré au système. Par contre, c'est le contrôleur de contrats qui est étendu pour initialiser et piloter les négociations atomiques relatives aux contrats non satisfaits qu'il gère. Pour cela, il va créer ou utiliser des nouveaux éléments du modèle de négociation qui seront implémentés sous la forme d'objets ou de composants définis dans les membranes des composants. De plus, le système de contrats donnera au système de négociation les accès aux contrats (clauses, participants, responsabilités des participants) pour qu'ils puissent être manipulés. Afin de faciliter le couplage avec le système de négociation, des *adaptateurs* vers les objets clauses et contrat originel seront fournis par le système de contrats et seront utilisés par les processus de négociation. Ces objets permettront d'obtenir la spécification originelle, de la remplacer et aussi de reconstruire des objets de contrats cohérents. Il est à noter que les négociations atomiques ne portent uniquement que sur les clauses du contrat violé. Les accords construits dans les contrats ne sont pas des objets de négociation. En revanche, ils sont censés être recalculés et éventuellement re-vérifiés en sortie de négociation atomique pour vérifier la validité du contrat original compte tenu des éventuelles modifications sur la clause négociée effectuées lors de la négociation. L'interfaçage avec les objets dans *ConFract* est précisé dans une section suivante.

Intégration dans le cycle de vie

Le processus de négociation démarre lorsque la vérification, par le gestionnaire de contrats d'une clause d'un contrat se révèle fausse. Le système de négociation effectue alors les opérations suivantes :

1. Isolation des parties des composants et des contrats impactés. Au niveau du système de contrats, une image du contrat existant nommé est formé à partir du contrat original. Cette image figée du contrat original n'observe plus l'exécution, ni les modifications qui interviennent sur les composants et les contrats. En particulier, cet objet contient aussi une copie des clauses sur lesquels vont porter les négociations atomiques. Ces dernières seront alors récupérés et adaptés pour devenir des objets de la négociation. Au niveau du système de composants, le fil d'exécution qui a déclenché la violation est bloqué avant de démarrer les négociations.

   Il s'agit d'une condition nécessaire et de l'hypothèse minimale que peut exiger le processus de négociation avant de démarrer. D'autres éléments du système peuvent nécessité d'être isolés, mais comme on ne peut pas, *a priori* et dans le cas général, identifier ces éléments et évaluer leurs

éventuelles influences sur le processus de négociation, le modèle de négociation ne définit donc pas de stratégies d'isolation supplémentaires. En revanche, l'intégration du modèle de négociation dans *Fractal* offre la possibilité d'effectuer des traitements arbitraires en pré ou post négociation par l'utilisation de crochets (ou *hooks*), un tel crochet avant pourra alors être utilisé pour intégrer une stratégie d'isolation dédiée, à effectuer avant le démarrage normal du processus de négociation.

2. Préparation des négociations atomiques en :
   – récupérant des éléments des contrats (clauses, responsables) ;
   – utilisant des éléments préalablement intégrés aux composants (capacités de négociation, règles d'interactions) ;
   – récupérant des éléments propres à la négociation (références aux parties négociantes, support pour la propagation de la négociation) ;
3. Construction et exécution des négociations atomiques ;
4. Re-synchronisation avec les parties des composants et contrats préalablement isolés.

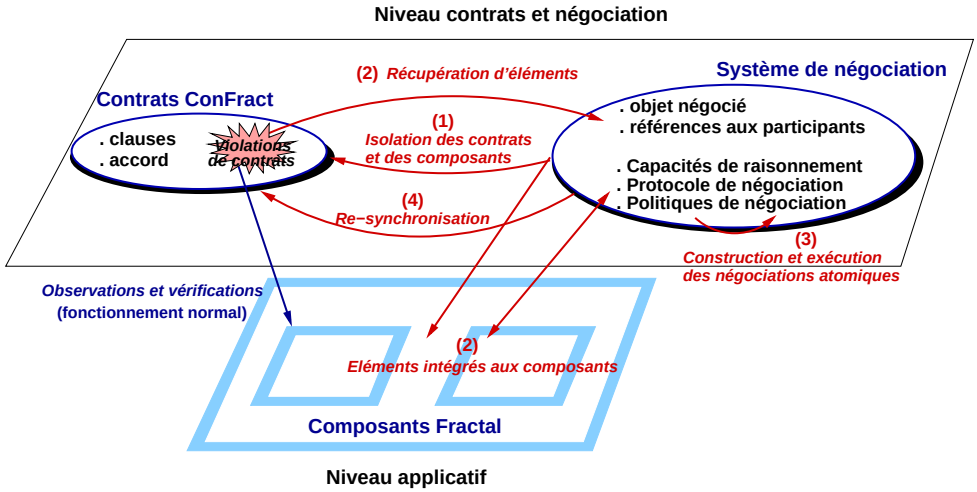


FIGURE *8.1 – Vue macroscopique de l'intégration de la négociation vis-à-vis des composants et des contrats*

8.2 Architecture

8.2.1 Choix de conception

Les négociations atomiques regroupent les différents éléments de base nécessaires à toute négociation et s'attachent chacune à rétablir la validité d'une clause de contrat en échec. Pour chaque négociation atomique, le modèle de négociation créé une représentation des parties négociantes en récupérant, de chaque clause en échec, le contrôleur de contrats qui gère le contrat concerné ainsi que les composants responsables, sur la base de leur responsabilités attribuées par le système *ConFract*. Ces parties négocient alors sur une clause de contrats en interagissant selon un protocole de négociation basé sur le *Contract-Net Protocol* (CNP) étendu. Pour cela, des capacités de négociation sont préalablement intégrés aux composants, et ces derniers, compte tenu de leur rôle d'initiateur ou de participant, exposent des interfaces leur permettant de conduire la négociation ou d'être consultés pendant celle-ci, selon les règles du protocole CNP.

Eléments réifiés

La figure 8.2 présente une vue simplifiée des principaux éléments réifiés dans le système de négociation. L'élément central est l'objet *AtomicNegotiation* qui regroupe l'ensemble des éléments requis pour négocier. Il se compose du *NegotiationObject* sur lequel porte la négociation, des parties négociantes *NegotiationParty* qui vont explicitement interagir au cours du processus de négociation selon les règles d'interactions du CNP, et du *NegotiationPolicy* qui oriente la négociation. A partir de cette première modélisation, il nous faut déterminer ce qui peut être représenté par des composants ou par des objets. Il est à noter que, bien qu'il représente un élément de base du modèle de négociation, le protocole de négociation n'est pas réifié puisqu'il détermine plutôt le comportement des parties négociantes en elles-mêmes. Cela est davantage décrit dans la suite de ce chapitre lors de la description de l'architecture des parties négociantes (cf. section 8.2.3).

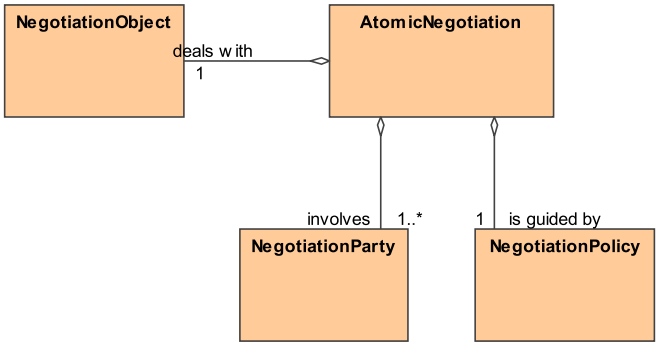


FIGURE 8.2 – *Modélisation des éléments de la négociation atomique*

Architecture en composants

Comme la version 2.5 de *Julia* introduit la possibilité d'implémenter les membranes des composants sous formes d'objets purs, mais aussi sous la forme de composants [12], nous faisons le choix d'élaborer une architecture à base de composants pour le système de négociation. La motivation première de ce choix réside dans le fait de pouvoir appliquer au système de négociation les qualités usuelles d'une approche à composants, avec en particulier une architecture explicite et des possibilités d'introspection et de reconfiguration dynamique des différents éléments du système de négociation. Ainsi, les membranes des composants métiers existants sont reconfigurées dynamiquement et leur architecture est modifiée pour ajouter l'ensemble des éléments nécessaires à la négociation, eux-mêmes architecturés sous la forme de composants. Par ailleurs, la seconde motivation consiste à exploiter l'architecture à base de composants *Fractal* du système de négociation pour appliquer, sur ce dernier même, les contrats ainsi que les mécanismes de négociation de *ConFract* (cf. page 95).

L'architecture du système de négociation proposée se base sur les choix suivants :

– la négociation atomique est modélisée à part entière en tant que composant. Le composant correspondant encapsule tout le processus de négociation. Il est construit par une *usine*, il s'initialise et s'exécute. Par ailleurs, comme la négociation atomique est *rattachée* au composant initiateur (c'est son contrôleur de contrats qui vérifie le contrat, démarre la négociation atomique...), le composant

12. http ://fractal.objectweb.org/current/doc/javadoc/julia/overview-summary.html#9

de négociation atomique est effectivement contenu dans la membrane du composant qui veut initier une négociation atomique (par exemple, dans la membrane d'un composant dont le contrôleur de contrats détecte une violation de contrats et souhaite déclencher une négociation atomique) ;
– comme notre modèle fait explicitement interagir des composants sous la forme de parties négociantes, chaque partie négociante du CNP est représentée par un composant. Ces composants représentent alors chacun une entité à part entière, autonome dans la négociation et potentiellement distribuée. De plus, chaque composant qui modélise une composant métier qui négocie réside dans la membrane de son composant métier et est aussi inclus dans le composant de négociation atomique par partage des instances maîtres ;
  Les deux éléments précédents – négociation atomique et parties négociantes – sont implémentés sous la forme de composants car ils représentent, du point de vue du système de négociation, les entités principales et stables de la négociation qui, respectivement, encapsule et fournit les services pour piloter une négociation, et implémentent le processus de négociation.
– le protocole de négociation définit essentiellement les règles d'interactions entre les parties négociantes – initiateur et participants. De ce fait, il n'est alors pas modélisé en tant que composant à part entière, mais il se retrouve à la fois dans la logique de pilotage de l'initiateur et dans les primitives de communication des participants permettant ainsi à l'initiateur de communiquer avec ces derniers ;
– l'objet de la négociation n'est pas réifié non plus en tant que composant. En effet, comme la négociation atomique porte sur cet objet de négociation, ce dernier ne constitue pas une entité métier de la négociation qui effectue ou requiert des traitements spécifiques, mais il fait plutôt partie du flux d'informations échangés entre les différents composants métiers de la négociation atomique (composants de parties, politiques de négociation...).
– enfin, des usines dédiées sont définies pour construire les composants qui modélisent une négociation atomique, et pour instancier dynamiquement chaque type de partie négociante.

8.2.2 Eléments récupérés des contrats

Objet de la négociation. Comme le rétablissement d'un contrat (objet *Contract*) se fait itérativement en gérant chaque clause violé du contrat, l'objet de négociation d'une négociation atomique est construit à partir de la copie de la clause originale. Pour découpler les objets manipulés par la négociation de ceux des contrats, un *NegotiationObject* est ainsi construit comme un *adapteur* vers la clause du contrat et les opérations de son interface *INegotiationObject* permettent de : *(i)* récupérer une description unique de la clause négociée du contrat, *(ii)* récupérer ses participants, *(iii)* modifier certains termes de la clause négociée, *(iii)* retirer ou remplacer entièrement la clause négociée, et *(iv)* effectuer de nouvelles vérifications de la clause négocié en simulant des observations du système contraint.

Références aux composants responsables. Les parties qui vont interagir lors des négociations atomiques sont les composants. Toutefois, comme le modèle de négociation fait intervenir les composants justement à partir de leurs responsabilités attribuées par le système *ConFract*, les références aux parties négociantes sont récupérées, non pas du système de composants, mais de l'objet de négociation *NegotiationObject* précédent. Ainsi, pour chaque clause négociée, les références de l'ensemble des participants ayant des responsabilités sont récupérées et ces derniers deviennent les parties négociantes des négociations atomiques.

En revanche, pour agir activement au cours des négociations atomiques, les parties négociantes devront être dotées de certaines capacités de négociation. Ces capacités sont internes à chaque composant et potentiellement ouvertes. Au minimum, les composants doivent avoir des capacités pour raisonner (« *quoi faire ?* ») et pour communiquer (« *comment discuter ?* »). Les capacités de raisonnement sont

écrites lors de la phase de configuration des composants et sont chargées en mémoire à l'initialisation du processus de négociation, pour instancier les composants qui représentent les parties négociantes. A l'exécution, et entre deux négociations actives, les capacités de négociation des composants peuvent aussi être modifiées. Celles-ci seront détaillées dans la suite.

8.2.3 Modélisation en Fractal des éléments

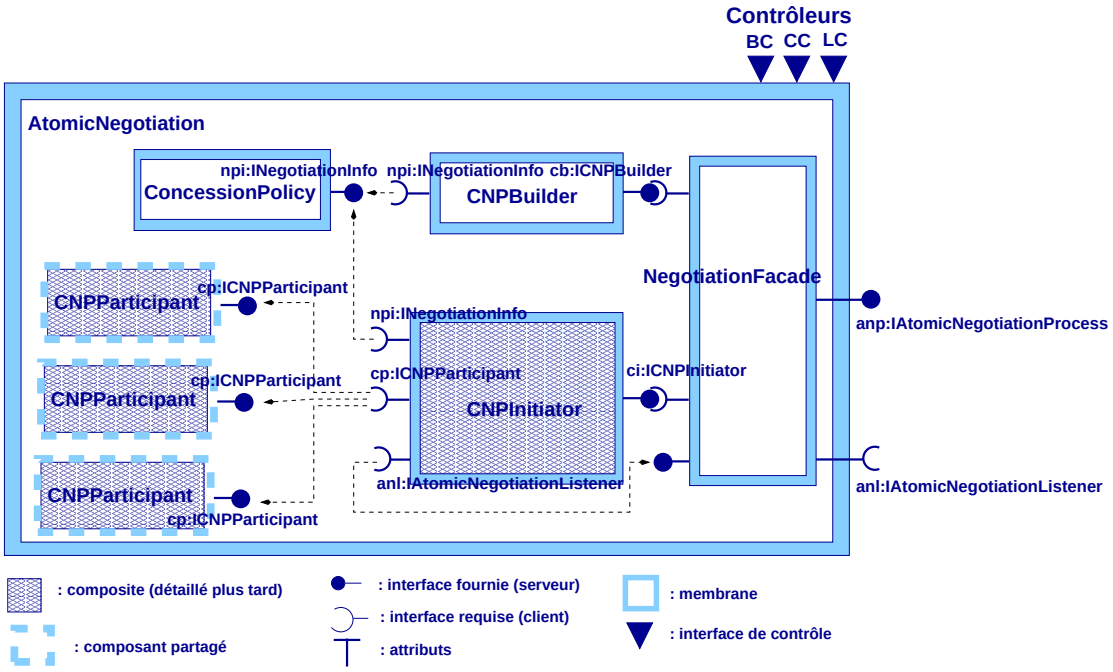
L'architecture *Fractal* des entités du système de négociation définit des composants à plusieurs niveaux. Une négociation atomique regroupe les éléments minimaux nécessaires pour négocier, et implémente le processus exécutable de la négociation. Des composants qui correspondent aux parties négociantes (initiateur et participants) sont explicitement définis. En particulier, le processus de négociation est piloté par l'initiateur du CNP, et toutes ces parties négociantes (initiateur et participants) sont nécessairement munies de capacités internes pour pouvoir interagir lors de la négociation selon les règles définies par ce protocole. De plus, diverses *informations* en terme de participants consultés par l'initiateur, ainsi que de type de négociation effectuée, viennent orienter le déroulement du processus de négociation.

Architecture de la négociation atomique

Une négociation atomique est modélisée par un composant composite (`AtomicNegotiation`). D'un point de vue externe, le composant `AtomicNegotiation` propose une interface `IAtomicNegotiationProcess` qui constitue le point d'entrée de la négociation atomique. Elle permet d'initialiser une négociation atomique (méthode `init(INegotiationObject)`), et d'exécuter (méthode `execute(INegotiationObject)`) le processus de négociation. Par ailleurs, pour pouvoir informer le composant client de la négociation atomique du résultat de la négociation, ce composant expose aussi une interface requise `INegotiationOutcomeListener` et agit en tant que source émettrice de l'événement `NegotiationOutcomeEvent` à l'écouteur client de la négociation atomique.

D'un point de vue interne, ce composant encapsule toute la logique du processus de négociation. Pour cela, comme la négociation s'appuie aussi sur des informations déterminées dynamiquement, ce composant `AtomicNegotiation` requiert d'être initialisé avant d'être exécuté. Par ailleurs, comme notre modèle fait explicitement interagir des composants sous la forme de parties négociantes, alors chaque partie négociante du CNP est représentée par un composant, et l'intérieur du composant `AtomicNegotiation` est alors modélisé de la façon suivante (cf. Fig. 8.3) :

- le sous-composant central est le composant `CNPInitiator`. Il modélise l'initiateur du CNP et, compte tenu de son rôle dans le CNP, il prend en charge le pilotage du processus de négociation, au travers de son interface offerte `ICNPInitiator` connectée au pendant requis du composant `NegotiationFacade` lequel interface et pilote l'ensemble des fonctions d'initialisation et d'exécution dans l'intérieur du composant `AtomicNegotiation`.
  Le composant `CNPInitiator` expose aussi une interface requise `ICNPParticipant` de type collection qui permet de le connecter à l'ensemble des participants CNP avec lesquels il va interagir lors de la négociation. Pour cela, ces connexions sont dynamiquement établies par le composant `CNPBuilder` (méthode `build()` invoquée par le composant `NegotiationFacade`) lors de l'initialisation de l'`AtomicNegotiation` (méthode `init` de l'interface `IAtomicNegotiationProcess`). Pour cela, le composant `CNPBuilder` utilise la liste des participants récupérée par son interface `INegotiationInfo` (méthode `getNegotiationParticipants`), et il effectue, en s'appuyant sur les interfaces contrôles de son englobant, des opérations de métamanipulations pour instancier les composants `CNPParticipant` et connecter leurs interfaces avec celle du `CNPInitiator`.



Interfaces

```
interface INegotiationObject{

 String getLabel();
 Collection<Participant> getParticipants();
...
}
```

```
interface INegotiationInfo{

 Collection<Participant> getNegotiationParticipants(
                         INegotiationObject o);
 String getNegotiationMessageType();
...
}
```

```
interface ICNPParticipant{

 boolean isNegotiable(String negoObjectLabel,
                      String negoMessageType);
 INegotiationProposal getProposal(String negoObjectLabel,
                                  String negoMessageType);
 boolean finalize(INegotiationProposal proposal,
                  String negoMessageType);
...
}
```

```
interface IAtomicNegotiationProcess{

 boolean init(INegotiationObject o);
 void execute(INegotiationObject o);
...
}
```

```
interface ICNPInitiator{

 void execute(INegotiationObject o);
...
}
```

```
interface INegotiationOutcomeListener{

 void notifyNegotiationOutcome(negotiationOutcomeEvent e);
...
}
```

```
interface ICNPBuilder{

 void build(String initiatorCName, INegotiationObject o);
...
}
```

FIGURE 8.3 – *Modélisation en Fractal d'une négociation atomique*

Une fois initialisé, le composant `CNPInitiator` exécute le processus de négociation selon les étapes du protocole du CNP (méthode `execute` de l'interface `ICNPInitiator`), et renvoie le résultat du processus par le biais de son interface `INegotiationOutcomeListener`. Dans la figure 8.3, les informations récupérées par le `CNPBuilder` sont fournies par le composant `ConcessionPolicy`, et les interactions entre ces deux composants sont ici volontairement décrites grossièrement, car le composant qui fournit l'interface serveur `INegotiationInfo` (ici `ConcessionPolicy`) dépend de la politique choisie et déterminé par un paramètre lors de la construction de la négociation atomique, selon la politique de négociation choisie. Les sections 8.3.1 et 8.3.2 décrivent plus en détails la construction de la négociation atomique, et les interactions entre `CNPBuilder` et ce composant.

– les autres sous-composants `CNPParticipant`, représentent chacun un participant négociant. Ils sont consultés par le `CNPInitiator` sur la base du CNP et à partir des primitives de communication exposées dans l'interface `ICNPParticipant`. Ces composants `CNPParticipant` sont construits dynamiquement et inclus dans le composant `AtomicNegotiation` par partage des instances maîtres résidant dans la membrane de chaque composant métier auxquels ils sont individuellement associés. A l'inverse, le composant de négociation atomique `AtomicNegotiation` et le composant `CNPInitiator` sont effectivement contenus dans la membrane du composant qui déclenche la négociation atomique.

La figure 8.4 donne une vue d'ensemble des différents éléments définis, ainsi que de leur localisation vis-à-vis des membranes des composants impactés.

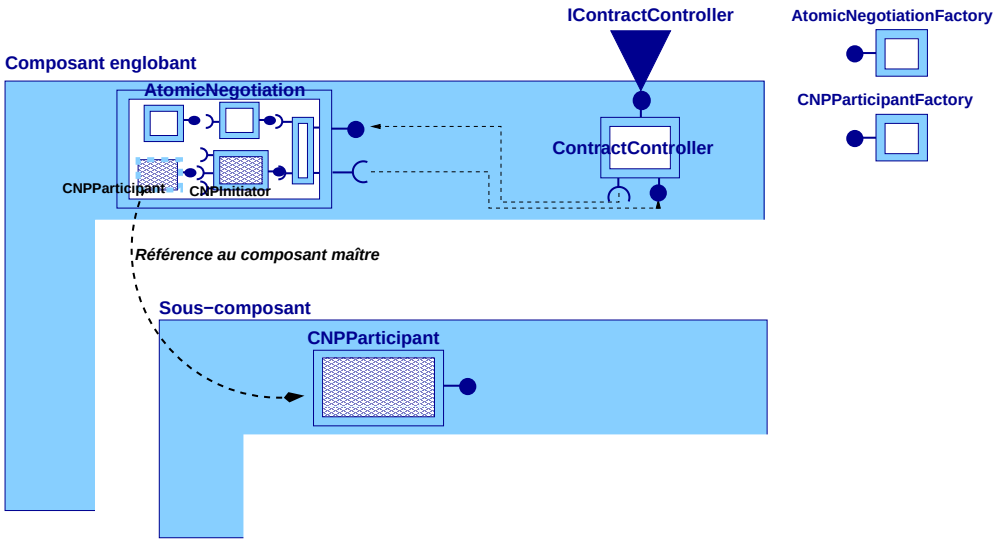


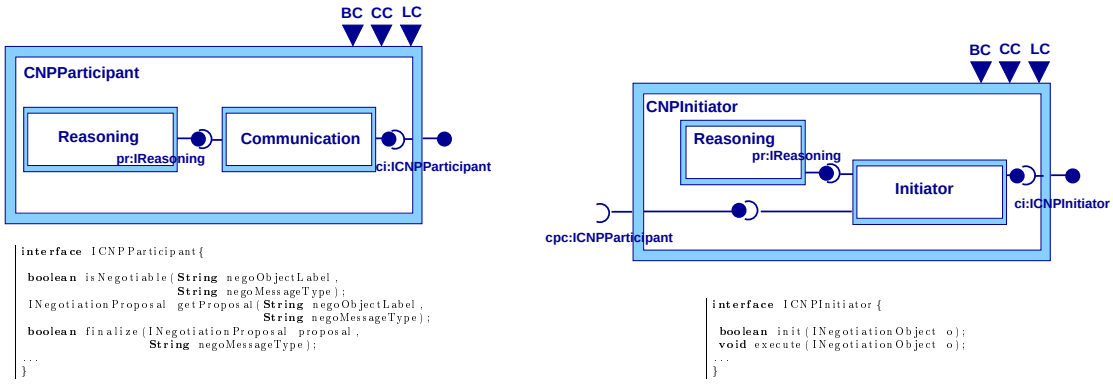
FIGURE *8.4 – Localisation des éléments des négociations atomiques dans les membranes*

Architecture des parties négociantes

Chaque partie négociante est modélisée par un composant Fractal qui expose une interface lui permettant d'être consultée sur la base du CNP. Pour remplir le rôle de l'initiateur du CNP, le composant associé `CNPInitiator` offre l'interface `ICNPInitiator` dont les méthodes `init` et `execute` permettent respectivement d'initialiser ses participants, et de démarrer le processus de négociation. Pour pouvoir interagir avec ses participants, le composant initiateur doit clairement avoir les références vers ces derniers, et son interface requise `ICNPParticipant` de cardinalité collection lui permet d'être connecté

à une ou plusieurs interfaces serveur `ICNPParticipant` des composants participants. Ces connections sont établies par le composants `CNPBuilder`. Les composants qui modélisent les participants du CNP exposent l'interface `ICNPParticipant` dont les méthodes `isNegotiable`, `getProposal` et `finalize` sont utilisées par l'initiateur pour que ce dernier les consulte.

Du point de vue interne, comme les composants doivent être munis de capacités pour pouvoir interagir dans la négociation, nous faisons le choix de représenter ces composants de parties sous la forme de composites constitués de sous-composants qui modélisent leurs capacités internes (cf. Fig. 8.5). Ainsi, les parties négociantes qui implémentent un participant du CNP (`CNPParticipant` à la figure 8.5a) contiennent au moins les deux sous-composants `Reasoning` et `Communication` qui encapsulent, respectivement, la stratégie de raisonnement (négociabilité, propositions...) et les capacités de communication (primitives de communication pour interagir) . Le composant qui modélise l'initiateur du CNP (`CNPInitiator` à la figure 8.5b) contient, quant à lui, deux sous-composants : `Initiator` qui contient la logique de pilotage selon le CNP et `Reasoning` qui encapsule la stratégie de l'initiateur au cours de la négociation (seuil de négociabilité, poids relatifs des participants...) et sont construits à partir des capacités de raisonnement fournies lors de la phase de configuration.



(a) Composant participant du CNP

(b) Composant initiateur du CNP

FIGURE 8.5 – *Modélisation en Fractal des parties négociantes*

8.2.4 Politiques de négociation

Le système de négociation définit actuellement deux types concrets de politique de négociation, par concession et par effort. Ces politiques sont chacune modélisées par un composant particulier `ConcessionPolicy`, pour la politique par concession, et `EffortPolicy`, pour la politique par effort. Ces deux composants implémentent l'interface `INegotiationInfo` et fournissent les informations suivantes :

- ils déterminent les composants responsables pour un objet de négociation donné (méthode `getNegotiationParticipants()`). Le composant `ConcessionPolicy` rend les références aux composants bénéficiaires alors que `EffortPolicy` rend la référence au composant garant ;
- ils définissent le type des messages de négociation (méthode `getNegotiationMessageType()`). Le composant `ConcessionPolicy` rend le type "message par concession" alors que le composant `EffortPolicy` rend le type "message par effort". Cette information est utilisée, lors de l'exécution de la négociation, par le `CNPInitiator` pour indiquer aux participants du CNP avec lesquels il interagit, le type de messages qu'ils échangent.

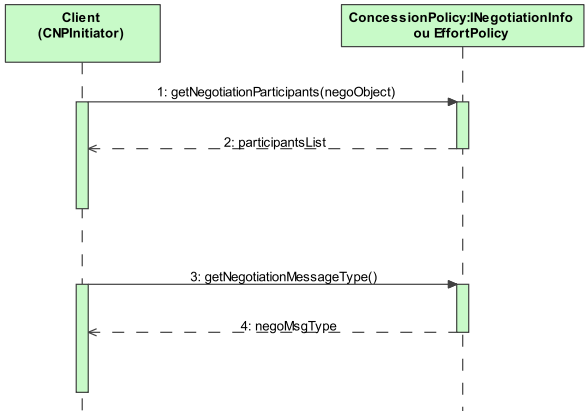


FIGURE 8.6 – *Utilisation des composants de politique de négociation*

Les parties qui vont interagir lors des négociations atomiques sont les composants. Toutefois, comme le modèle de négociation consiste justement à faire négocier des composants précisément déterminés à partir de leurs responsabilités ou contributions dans l'objet négocié, les références aux parties négociantes doivent être fournis au composant initiateur par des entités dédiées capables de raisonner sur les contrats ou l'architecture des composants. Le système de négociation définit deux types d'entités. Ce sont soit les composants de politique précédents, dont les logiques internes sont définies statiquement (le composant de politique par concession rend les références aux composants bénéficiaires, le composant de politique par effort rend les références au composant garant, etc.), soit une autre entité *particulière*, capable de fournir des informations compositionnelles en raisonnant sur les propriétés négociées. Cette dernière entité est le *gestionnaire de propriétés compositionnelles* entité, définie précédemment et qui fournit diverses informations décrivant la décomposition structurelle de certaines propriétés compositionnelles des composants (cf. section 7.4, page 102).

Le choix de l'utilisation de l'un ou l'autre des composants se fait en fonction du type de négociation (négociation initiale ou propagée) et se définit par paramétrage lors de la construction du composant de négociation atomique. Au niveau de hiérarchie de la négociation initiale, c'est le composant de politique qui fournit la (première) liste des parties négociantes en exploitant les responsabilités de la clause négociée du contrat. En cas de propagation de la négociation, c'est le composant qui permet de raisonner compositionnellement sur les propriétés négociées qui fournira, à chaque niveau de hiérarchie, la liste des composants qui interviennent dans la décomposition des propriétés négociées.

8.3 Fonctionnement

8.3.1 Construction et initialisation de la négociation atomique

La construction de la négociation atomique s'effectue en deux étapes. Une première étape permet de construire partiellement le composant `AtomicNegotiation`. Une seconde étape est nécessaire pour instancier par des informations contextuelles les composants qui modélisent les parties.

Une usine dédiée `AtomicNegotiationFactory` permet de construire une négociation atomique (méthode `createAtomicNegotiation()`) (cf. Fig. 8.7). Cette méthode de création prend en paramètre une information qui décrit le type de négociation atomique à instancier, de façon à préciser le composant qui va fournir l'interface serveur `INegotiationInfo` et qui sera utilisé par le compo-

sant CNPBuilder (cf. Fig. 8.3). Ainsi, dans le cas d'une négociation *initiale*, l'usine crée le composite AtomicNegotiation avec un composant de politique (ConcessionPolicy ou EffortPolicy). Dans le cas d'une négociation "propagée", l'usine pour crée l'AtomicNegotiation avec le composant de gestionnaire de propriétés compositionnelles (CompositionalPropertyManager).

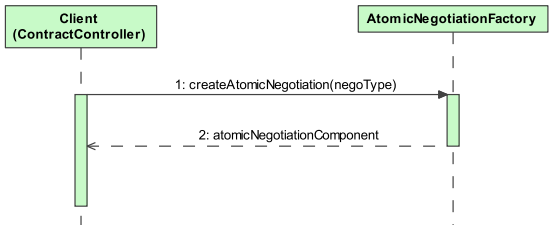


FIGURE 8.7 – *Construction de la négociation atomique (composant AtomicNegotiation)*

Une fois construit, le composant AtomicNegotiation est initialisé (méthode init() de IAtomicNegotiationProcess). L'implémentation de cette initialisation faite par le composant CNPBuilder consiste à affecter la liste des participants effectifs du CNP à l'initiateur CNPInitiator par des opérations de méta-manipulation effectuées en utilisant les interfaces de contrôles correspondantes de l'englobant AtomicNegotiation. Plus précisément, la liste des participants effectifs est récupérée, par le CNPBuilder, du composant qui implémente l'interface INegotiationInfo. Puis, avec les références de ces participants, les composants participants sont instanciés par une autre usine dédiée (CNPParticipantFactory) et partagées à la fois dans la membrane du composant participant (instance maître) et dans l'intérieur du composant AtomicNegotiation (instance esclave). Ce partage est utilisé pour à la fois rattacher les composants participants dans la membrane de leur composant, et les inclure dans le composant AtomicNegotiation. Enfin, les interfaces serveur ICNPParticipant des instances esclaves de ces participants sont connectées à celles requises du CNPInitiator. Une fois ses participants clairement connectés, le CNPInitiator est alors prêt à piloter la négociation atomique et celle-ci peut-être exécutée.

8.3.2 Exécution de la négociation atomique

La négociation atomique s'exécute par la méthode execute() de l'interface IAtomicNegotiationProcess. Son exécution déclenche celle du composant CNPInitiator (cf. Fig. 8.9). Dès lors, le composant CNPInitiator prend lui même en charge le processus de négociation, et il interagit avec les participants avec lesquels il a été connecté dans la phase ultérieure d'initialisation afin de négocier selon les trois phases du CNP (cf. Fig. 8.10).

- le composant CNPInitiator demande la négociabilité aux participants (méthode isNegotiable()) en passant en paramètre le label de la clause négociée, ainsi que le type de la politique courante (message de type concession ou effort) de façon à ce que les participants aient la connaissance du type de négociation sur lequel ils sont consultés ;
- il récupère au fur et à mesure les propositions des participants (méthode getProposals), et utilise son composant Reasoning et les interfaces de manipulation de l'objet négocié pour revérifier la clause modifié compte tenu des propositions des participants ;
- il renvoie un événement NegotiationOutcomeEvent qui décrit le résultat de la négociation.

La phase de construction sert à paramétrer une négociation atomique selon une politique de négociation donnée. Une fois construit, le composant de négociation est alors initialisé en utilisant la politique définie à sa construction ainsi que certaines informations dynamiques qui lui sont fournies (objet de né-

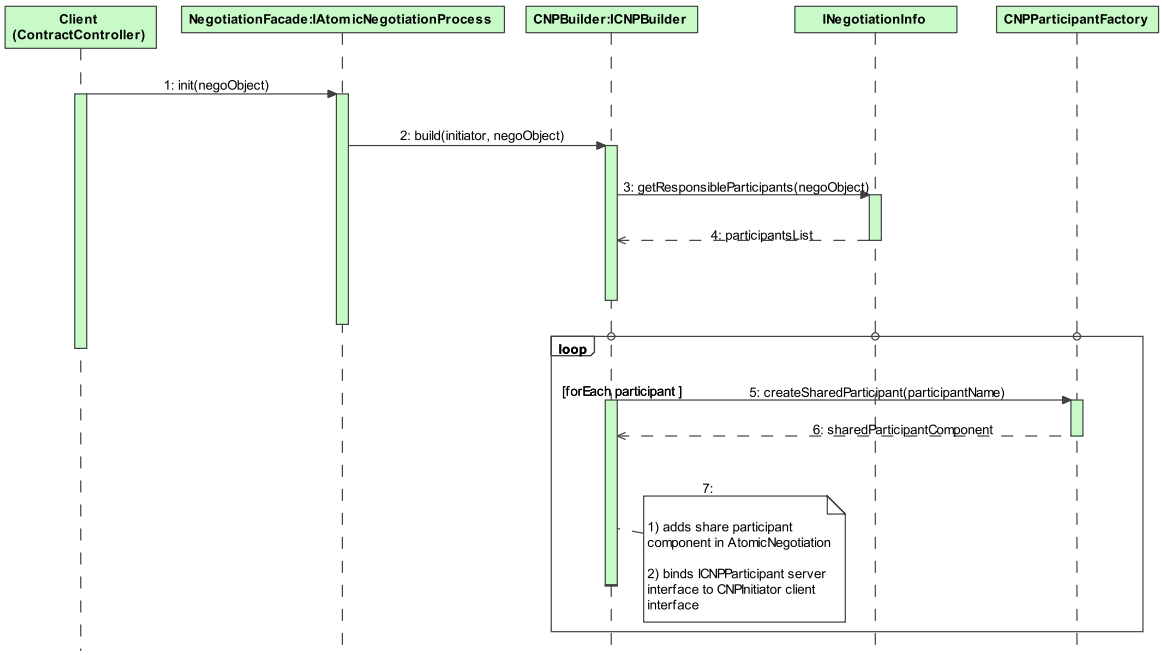


FIGURE *8.8 – Initialisation de la négociation atomique (composant AtomicNegotiation)*

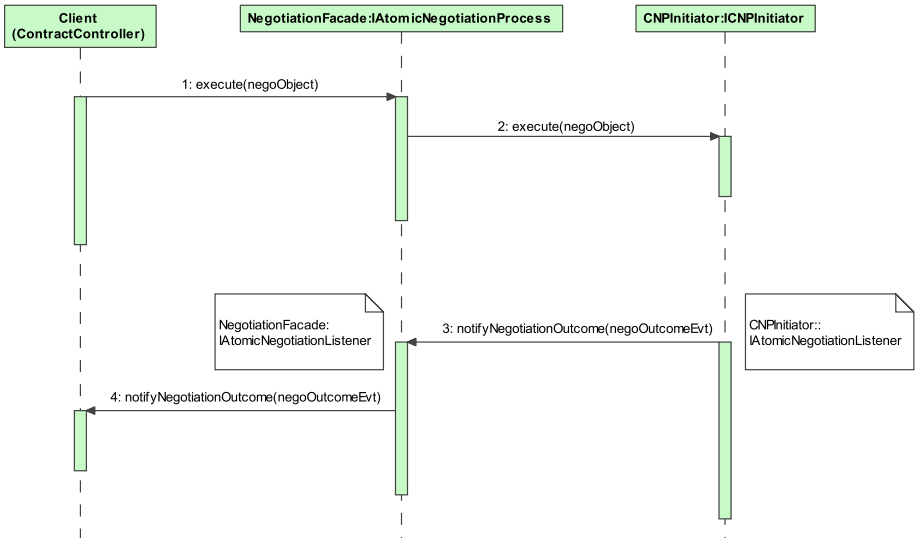


FIGURE *8.9 – Exécution de la négociation atomique (composant AtomicNegotiation)*

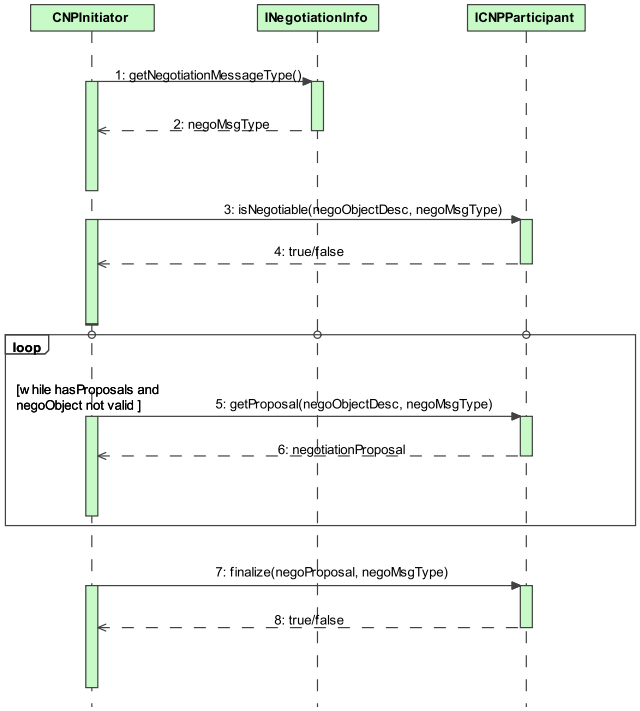


FIGURE *8.10 – Interactions entre initiateur et participants du CNP*

gociation notamment). Dès lors, les phases de construction et d'initialisation déterminent entièrement tous les éléments d'une négociation atomique et le composant de négociation atomique n'est valable que pour un contexte de violation et un ensemble de parties négociantes donné. Le fonctionnement de base actuel d'une négociation atomique consiste donc à exécuter une séquence de construction, suivi d'une initialisation puis d'une exécution (`create ; init ; execute`). Pour une initialisation donnée, il n'est pas intéressant d'exécuter plusieurs fois une négociation atomique (`init ; execute`$\star$), car celle-ci ne varie pas puisqu'elle est isolée à la fois des contextes de contrats et de composants. De plus, comme le paramétrage de la négociation atomique est déterminé lors de sa construction, les répétitions des séquences d'initialisation et d'exécution (`(init ; execute)`$\star$) ne sont intéressantes que si une nouvelle construction est effectuée.

Après construction, les négociations atomiques sont exécutées et travaillent sur l'objet cloné, *NegotiableClause* correspondant. Les interactions des parties sont régies par le protocole de négociation mais leur exécution dépend essentiellement des capacités de négociation des parties consultées. Une fois que le processus de négociation aboutit à un *NegotiableClause* acceptable, ce-dernier est utilisé pour mettre à jour la clause du contrat original. Il s'agit essentiellement de copier les modifications effectuées sur l'objet négocié vers celui du contrat original afin que le système de contrats puissent les prendre en compte. La partie du système de composants qui avait été isolée à l'entrée de la négociation est alors redémarrée, et le cycle d'exécution normal du système et des contrats reprend. Si la négociation atomique ne permet pas de rétablir la validité de l'objet négocié, alors la négociation s'achève par un échec signalé par une exception.

Enfin, pour préserver les règles de visibilité des composants, la portée spatiale des négociations atomiques est restreinte à des parties négociantes qui ne sont liées par au plus qu'un même niveau de hiérarchie des composants.

8.3.3 Propagation d'une négociation atomique

Comme la portée spatiale des négociations atomiques conserve le niveau d'encapsulation des composants, une nouvelle négociation atomique doit être construite. Un nouveau composant `AtomicNegotiation` est donc construit dans la membrane de tout composant qui initie ou propage la négociation. Au niveau de hiérarchie de la violation initiale, le composant de négociation atomique est hébergé dans la membrane du composant contenant le contrôleur de contrats qui démarre la négociation atomique initiale. Lorsqu'une négociation se propage, le nouveau composant `AtomicNegotiation` est construit dans la membrane du composant participant qui propose la propagation de façon à former de nouveau les participants de la négociation atomique. Ce nouveau composant est construit et initialisé par le composant participant, car c'est ce dernier qui a proposé la propagation sous la connaissance des nouvelles parties négociantes. Ce composant participant est aussi le client de la nouvelle négociation atomique. De ce fait, et étant aussi participant de la négociation atomique initiale, il fait naturellement le lien entre les deux négociations initiale et propagée (cf. Fig. 8.11). Ce composant peut alors recevoir les résultats de la négociation propagée et les retransmettre au composant initiateur du niveau du dessus. Par la suite, ce même schéma de construction est appliqué, en cas de nouvelle propagation.

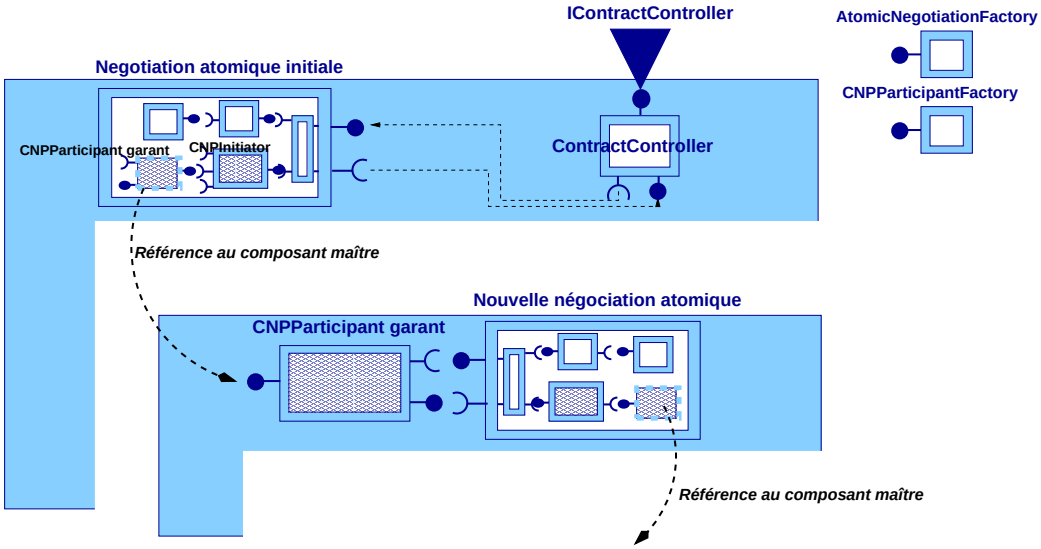


FIGURE 8.11 – *Eléments des négociations atomiques propagées*

Il est à noter que comme le modèle *Fractal* n'autorise pas la mutabilité des types de composants, un composant participant garant doit d'emblée exporter les interfaces qui lui permettent d'être client d'une éventuelle nouvelle négociation atomique, même s'il se trouve que par la suite, au cours d'une négociation atomique, aucune négociation n'est effectivement propagée (propagation non proposée par le composant participant garant, propagation non acceptée par l'initiateur de la négociation). Ces interfaces ont ainsi un caractère optionnel.

9 Évaluation

Dans ce chapitre, nous effectuons une évaluation du modèle de négociation proposé. Dans un premier temps, nous effectuons une évaluation qualitative du modèle et du processus de négociation, et nous discutons leurs limites. Puis, nous présentons quelques éléments méthodologiques relatifs à l'intégration du modèle dans le cycle de vie, à ses points de variabilité et à la caractérisation de ces derniers. Enfin, la dernière partie de ce chapitre présente les premiers résultats de l'évaluation de performances des mécanismes de négociation.

9.1 Évaluation qualitative

Nous reprenons ici les qualités déterminées dans le chapitre 5 pour effectuer une évaluation qualitative des propositions.

9.1.1 Qualités du modèle

Nous revenons ici sur les qualités attendues du modèle de négociation.

Extensibilité. Des extensions du modèle sont possibles au niveau des types de contrat supportés, du moment qu'ils fournissent une expression des responsabilités associés aux composants participants ou à d'autres entités logicielles. De même, d'autres politiques de négociation ou des combinaisons des politiques proposées semblent potentiellement possibles (protocole donnant/donnant par exemple), bien que le travail pour les définir semble encore conséquent.

Applicabilité. Nous effectuons ici une analyse des éléments du modèle qui doivent être définis afin de réutiliser tout ou partie de celui-ci dans un autre contexte :

- Les contrats à négocier partent d'une décomposition en clauses, chacune établissant des responsabilités sur l'architecture, qui sont exploitables par les politiques.
- Les dépendances avec l'architecture sont diverses : Les entités logicielles doivent être équipées des capacités à négocier ; les responsabilités leur sont affectées ; des informations supplémentaires pour raisonner sur l'architecture peuvent être exploitées par des politiques de négociation — formules compositionnelles et plus généralement formules permettant la déduction —.

– Le processus de négociation doit être piloté par une entité ayant accès aux contrats et à l'architecture de façon à s'organiser selon le CNP, c'est-à-dire avec une cardinalité 1-N entre l'initiateur et les participants.

L'applicabilité du modèle dans le cadre d'une plate-forme autre qu'à base de composants, pourra être déterminée par une étude complète de la plate-forme de projection.

Adéquation à Fractal et ConFract. Le modèle et les politiques associées suivent l'organisation hiérarchique des composants. La relation entre l'initiateur de la négociation et les contractants est fortement liée à la relation entre un composite et ses sous-composants. Le gestionnaire de contrats confine *a priori* la négociation à un niveau de hiérarchie, préservant l'encapsulation. En effet, comme il est situé sur le composant englobant, il consulte les composants bénéficiaires ou garant qui sont soit lui-même soit ses sous-composants. Par ailleurs, lorsque la négociation est amenée à se propager, une nouvelle négociation atomique est construite et celle-ci se déroule selon les règles d'encapsulation des composants (voir chapitre 8).

9.1.2 Qualités du processus

Nous discutons maintenant les qualités du processus de négociation.

Simplicité. Cette qualité est assurée par l'utilisation sous-jacente du CNP, qui définit un modèle d'interactions simple et proche de la réalité des négociations. Les processus de négociation sont définis sur un noyau minimal constitué des objets, des parties et du protocole de négociation, auxquels viennent se greffer les politiques de négociation pour mettre en œuvre complètement les processus de négociation. Ainsi, la complexité du processus de négociation résulte directement de celle des politiques et comme le modèle est extensible, des politiques plus complexes pourraient toujours être intégrées.

Raisonnement privé. Les composants négocient à travers le protocole fourni, mais rien ne permet de connaître leur raisonnement, hormis en observant leurs propositions successives lors des négociations. Ceci assure un raisonnement privé de même nature que dans les systèmes multi-agents.

Ouverture vs. Contrôle. L'ouverture du processus est relativement importante. Par exemple, dans la politique par concession, les modifications sur les formules ou les reconfigurations de paramètres peuvent entraîner, dans un modèle aussi ouvert que *Fractal*, de nombreuses incohérences. Contrôler *a priori* les modifications proposées par les parties afin d'assurer certaines propriétés de cohérence est, dans le cas général, impossible. Il faudrait fortement réduire la portée des modifications et même le formalisme utilisé afin de vérifier que les modifications aient une portée contrôlée sur les entités ou les formules impactées. Même avec ces affaiblissements, l'ouverture complète de *Fractal* peut faire qu'une simple modification d'attribut sur un composant peut être liée à une implémentation quelconque, modifiant de manière incontrôlée un ensemble de composants (mise à jour de composants, enchaînement de reconfiguration architecturales...). Nous préférons conserver toute la puissance aux modifications, afin d'avoir un grand pouvoir d'expressivité dans les négociations, tout en nous reposant sur les contrats eux-mêmes pour assurer des contrôles. En effet, les clauses de contrats modifiées sont réévaluées après négociations, et les contrats environnants restent actifs, assurant ainsi des contrôles locaux, et éventuellement plus globaux. En revanche, cette approche ne permet pas d'éviter une déstabilisation globale du système en cas de violation de contrats en cascade ou en cycle, lors de renégociations à différents endroits d'un système (cf. limites ci-après).

Stabilité et reproductibilité. Comme nous l'avons dit dans le chapitre 5, la reproductibilité du processus dépend fortement de l'environnement dans lequel interagissent les parties. Si on considère un environnement constant autour des parties, la seule chose qui peut influencer le processus de

négociation est la durée limite de réception des propositions et donc la possibilité que certaines propositions n'arrivent pas à l'initiateur. Les communications entre les participants et l'initiateur sont potentiellement distribuées et un tel scénario est donc envisageable.

9.1.3 Limites

La principale limite du modèle proposé concerne les contrôles suite aux négociations et la propagation des négociations à travers les composants. Comme nous l'avons discuté ci-avant, le contrôle du processus de négociation est limité à l'utilisation des contrats eux-mêmes pour vérifier la cohérence des propositions, et détecter les contrats remis en cause par les négociations. En revanche, rien ne permet de vérifier que des contrats renégociés ou des adaptations sur les composants n'entraînent des violations et renégociations en cascade. De même, lorsque des politiques de négociation peuvent amener à des propagations, comme dans le cas de la politique par effort, il n'est pas possible de déterminer *a priori* la terminaison d'une ou plusieurs propagations. Cela vient en particulier du fait qu'il n'y a aucune *inférence* ou *calcul de dépendances* entre des contrats ou entre les différentes actions d'adaptations potentielles des composants, qu'il serait en effet trop complexe et coûteux d'exprimer. De plus, comme nous l'avons précédemment souligné, l'ouverture de la plate-forme *Fractal* et ses fortes capacités réflexives empêche de garantir ou d'effectuer des vérifications *a priori* sur les différentes modifications proposées au cours des négociations. Rien ne peut empêcher cela *a priori* ; les politiques n'effectuent aucune vérifications dans ce sens et c'est à l'assembleur de s'assurer du comportement souhaité autant que possible.

Pour dépasser cette limite, une réponse possible consiste à paramétrer les processus et politiques de négociation par des contrats, puis d'utiliser les mécanismes de négociation eux-mêmes pour limiter la propagation et détecter les négociations en cascade non désirées. D'autres possibilités sont introduites dans les perspectives.

9.2 Aspects méthodologiques

Cette section vise à fournir quelques éléments méthodologiques pour faciliter l'utilisation du modèle de négociation dans la plate-forme *Fractal*/*ConFract*. Nous présentons ainsi rapidement comment les contrats et la négociation s'intègrent dans le cycle de vie des systèmes à composants, puis nous décrivons et caractérisons les points de variabilité du modèle de négociation.

9.2.1 Intégration dans le cycle de vie des composants

Vue d'ensemble sur le cycle de vie des systèmes à composants

En fonction de la richesse des plates-formes à composants, plusieurs étapes différentes dans le cycle de composants peuvent être considérées. Les plates-formes de composants les plus avancées sont celles qui maintiennent et permettent de manipuler les composants aux phases ultérieures au développement, lors de la configuration et de l'exécution, et permettent de les reconfigurer lors de l'exécution. Dans ces plates-formes riches, le cycle de vie usuel se décompose alors selon les phases suivantes :

1. Phase de développement/réutilisation : les composants sont obtenus soit par développements spécifiques, soit par réutilisation de composants existants (COTS) avec d'éventuelles adaptations ;
2. Phase d'assemblage et configuration : ils sont alors assemblés pour construire l'architecture globale de l'application. A ce moment, ils peuvent aussi être configurés pour prendre en compte les spécificités des autres composants de l'assemblage. Ces deux phases sont réitérées et l'architecture est progressivement raffinée jusqu'à ce que l'application soit complètement construite ;

3. Phase de déploiement : l'application est alors déployée et installée dans son environnement d'exécution ;
4. Phase de configuration de l'application : une fois l'environnement d'exécution connu et avant de démarrer l'application, certains composants de l'application peuvent une nouvelle fois être réglés pour, cette fois-ci, prendre en compte des facteurs relatifs à l'environnement et au contexte de déploiement ;
5. Phase d'exécution : l'application est démarrée et s'exécute ;
6. Phase de reconfiguration : en cours d'exécution, des reconfigurations de l'application peuvent être effectuées (reconfigurations de l'architecture, mises à jour de composants, réglages de paramètres divers...) afin de faire évoluer le système, ou de prendre en compte de nouvelles caractéristiques. Tout ou partie du système est alors arrêté pour permettre ces reconfigurations (cf. phase 4), et le système est redémarré par la suite.

Etapes supplémentaires pour l'intégration des contrats

L'intégration des contrats du système *ConFract* dans le cycle de construction et d'exécution précédent, modifie ce dernier de la façon suivante. En phase de développement ou d'assemblage du système (phase 2), les spécifications textuelles qui spécifient les propriétés sur les éléments d'architecture (interfaces et compositions des composants) sont écrites. En fin de configuration (fin de la phase 4), les contrats sont dynamiquement construits à partir des spécifications stockées, et ils sont armés. Par la suite, les clauses des contrats de configuration sont vérifiées à ce moment, alors que toutes les autres le sont lors de l'exécution du système (phase 5). Enfin, lors des reconfigurations du système (phase 6), les contrats impactés par ces reconfigurations sont aussi mis à jour.

Etapes supplémentaires pour l'intégration de la négociation

Comme nous l'avons vu précédemment, le modèle de négociation définit plusieurs éléments de base. Les objets de la négociation sont les clauses en échec des contrats, les parties sont les composants ayant des responsabilités dans ces clauses, et le protocole de négociation est inspiré du CNP. Ces éléments sont complètement définis lorsque la négociation cible la plate-forme *Fractal/ConFract*, et ils constituent les parties fixes des négociations atomiques, automatiquement déterminées par le système de négociation lui-même. En revanche, les éléments relatifs à "l'intelligence" de la négociation (modèles de décisions des parties, configuration diverses du processus de négociation...) ne sont pas fixés par le modèle, car ils dépendent des domaines d'applications et des logiques d'adaptation spécifiques que l'on souhaite fournir aux applications et à leurs composants. Ces éléments sont requis et doivent être fournis lors de la contractualisation des systèmes (négociabilité, propositions de modifications, poids d'importance...). Toutefois, bien qu'étant très importants, ils sont souvent très difficiles à définir, car cela suppose de connaître très précisément le contexte d'utilisation ainsi que l'évolution des systèmes contractualisés (cas de fonctionnement normal, cas de défaillances, actions de rattrapages à mener...). De ce fait, nous ne nous focaliserons pas sur la méthodologie pour élaborer les (bonnes) actions d'adaptations. En revanche, dans la suite de cette section, nous identifions et caractérisons les points de *variabilité* du modèle qui permettent de régler différemment le processus de négociation.

9.2.2 Points de variabilité du modèle

La modèle de négociation répartit le processus de décision sur trois niveaux différentes, du niveau le plus bas au niveau le plus haut, de la façon suivante :

– les capacités individuelles des composants ;

– les interactions des parties négociantes, définies par le protocole ;
– le pilotage des négociations atomiques par les politiques.

Le niveau le plus bas définit les capacités individuelles des composants qui peuvent proposer et effectuer diverses modifications dans leur portée. Moralement, ces capacités décrivent ce que chaque composant peut faire s'il est amener à négocier, et l'ensemble des capacités des parties qui vont négocier décrit l'espace des solutions possibles. Au niveau intermédiaire, le processus d'interaction, régi par le protocole de négociation, exploite les capacités des parties consultées. Ce processus d'interactions formalise le processus de décision en mettant en jeu par la négociation toutes les parties ayant des capacités, et en permettant d'en dégager une solution satisfaisante (ie. la modification proposée qui va revalider une clause). A ce niveau, le pilotage se fait essentiellement par le protocole de négociation défini, néanmoins, comme l'initiateur pilote les interactions, il peut aussi paramétrer de diverses façons son déroulement. Enfin, au niveau supérieur, ce sont les politiques de négociation qui décrivent le déroulement complet des négociations atomiques. Chaque politique fixe individuellement une orientation particulière aux négociations atomiques (responsabilités exploitées, caractéristiques souhaitées...), et les politiques peuvent être pilotées pour organiser le déroulement global des négociations.

Ainsi, chaque niveau permet d'intégrer plus ou moins d'intelligence au processus de négociation. Les composants dont on souhaite qu'ils négocient doivent être préparés et leurs capacités de raisonnements doivent être définis. De plus, le protocole de négociation et les politiques constituent aussi des éléments de pilotage de la négociation. La suite caractérise chacun de ses éléments.

9.2.3 Caractérisation des points de variabilité

Capacités des parties négociantes

Les capacités de raisonnement des parties négociantes reposent sur la définition des alternatives. Ainsi, pour chaque composant à qui l'on souhaite fournir des capacités de négociation, une liste d'alternatives doit être définie et lui être associée pour chaque politique et pour chaque clause dans laquelle le composant a un rôle à jouer. Ces listes d'alternatives sont définies au même moment où les spécifications sont écrites (phase 2) et elles contiennent l'ensemble des actions correspondantes à chaque politique. Celles qui correspondent aux parties impliquées sont chargées en mémoire lors de l'initialisation d'une négociation atomique, mais elles peuvent être modifiées entre deux négociations atomiques qui ne les utilisent pas.

Les tableaux 9.1 et 9.2 récapitulent pour chaque politique les actions possibles et leur signification du point de vue des composants responsables impactés.

Politiques de négociation

Les adaptations effectuées par le processus de négociation sont orientées par la politique de négociation choisie. Les deux politiques par concession et par effort ont été entièrement décrites et chacune d'entre elles peut être choisie par l'initiateur de la négociation au moment de former une négociation atomique. Pour guider le choix entre ces deux politiques, nous les caractérisons selon les critères suivants.

Confinement et propagation. Dans les deux politiques, les négociations atomiques sont confinées au niveau de hiérarchie de la violation et font intervenir des parties négociantes liés à un niveau de hiérarchie au plus. Dans la politique par concession, ces parties restent constantes alors que dans la politique par effort, comme une négociation atomique initiale peut se propager dans la hiérarchie, de nouvelles négociations atomiques peuvent se former impliquant alors des composants à des niveaux inférieurs. Cela permet en partie de préserver l'encapsulation des composants et les

Actions possibles	**Signification en terme de capacité des composants responsables**
Non négociable	Les bénéficiaires n'ont pas de capacités de négociation. La clause ou certains de ses termes ne sont pas modifiables
Maintien de la clause ('S-top')	Les bénéficiaires ont besoin de la clause. Celle-ci doit être maintenue en l'état car la contrainte formulée est importante
Retrait de la clause ('Release')	Les bénéficiaires acceptent de se passer de la clause. Celle-ci peut alors être retirée car la contrainte formulée bien qu'importante reste plutôt optionnelle
Modification de la clause ou de ses termes	Les bénéficiaires peuvent relâcher tout ou partie de la contrainte formulée. Celle-ci peut-être assouplie mais la contribution des composants bénéficiaires reste inchangée

TABLE *9.1 – Récapitulatif des actions pour la politique par concession*

Actions possibles	**Signification en terme de capacité du composant responsable**
Non négociable	Le composant garant n'a pas de capacité de négociation en terme de possibilités d'efforts ou de propagation
Actions d'effort sans propagation	Le garant peut effectuer un effort lui-même à son niveau mais ne peut pas propager. Il est responsable de l'implémentation de la propriété négociée
Propagation sans actions d'efforts	Le garant n'a pas la capacité d'effectuer un quelconque effort à son niveau mais il peut propager vers des sous-composants. Il est responsable de son assemblage et certains de ses sous-composants réalisent la propriété négociée
Propagation et actions d'efforts	Le garant peut à la fois proposer un effort et propager. Il peut définir un ordre de préférence ou alors c'est l'initiateur de la négociation qui organise ces deux capacités

TABLE *9.2 – Récapitulatif des actions pour la politique par effort*

niveaux de visibilité entre l'initiateur d'une négociation – le contrôleur de contrats du composant englobant – et les autres parties – le composant englobant ou des sous-composants de ce dernier.

Impact sur l'architecture. Dans la politique par concession, les actions d'adaptations effectuées sont plutôt légères et concernent les clauses des contrats, à fonctionnement et architecture des composants constants, soit des reconfigurations légères des attributs de composants. Dans la politique par effort, les actions sont plus ouvertes, et peuvent consister en des reconfigurations d'attributs, mais aussi en des codes d'adaptation dans la portée du composant garant.

Portée des actions. Dans chacune des deux politiques de négociation, comme les actions possibles sont proposées par les composants responsables impliqués, celles-ci sont limitées par la portée de ces composants. Dans la politique par concession, les modifications ne portent que sur les clauses de contrats, seuls éléments visibles des bénéficiaires. Dans la politique par effort, les actions effectuées le sont dans la portée du garant uniquement ou des composants contributeurs s'il y a propagation. Cette notion de portée des actions est importante. De la même façon que chaque négociation atomique a une portée spatiale restreinte au niveau de hiérarchie de la violation (cf. 1er point), les actions effectuées que l'on retrouve en sortie du processus de négociation sont proposées par les parties consultées. Elles sont donc de fait restreintes à la portée de ces dernières.

Validation par les contrats et non-déterminisme. De façon complètement orthogonale aux politiques de négociation, les actions de négociation effectuées sont validées par les clauses des contrats eux-mêmes. Les actions d'adaptation ne sont donc pas complètement déterminées à l'avance. Les moments de négociation qui correspondent aux moments auxquels les contrats sont violés, ne sont pas connus. De plus, les actions d'adaptations effectuées *in fine* ne sont pas pré-déterminées puisqu'elles résultent à la fois du du processus de négociation, lui-même paramétré par les points de variabilité évoqués précédemment (cf. section 9.2.2), et aussi du contexte de violation (les solutions sont vérifiées par évaluation et revalidation des contrats violés).

Actions en dehors du processus de négociation. Finalement, en dehors de tout processus de négociation, le modèle conserve la possibilité d'intégrer divers codes d'adaptations au niveau des contrôleurs de contrats. De tels codes peuvent être beaucoup plus riches et plus étendues (reconfigurations de composants, actions combinées...). Cependant, mais qui doivent cependant rester dans la portée du contrôleur de contrats qui gère la clause de contrat en échec (son composant porteur et ses sous-composants).

9.3 Premières évaluations expérimentales

9.3.1 Environnement

Dans cette section, nous présentons les résultats expérimentaux des premières évaluations du modèle de négociation. Nous n'évaluons pas ici l'efficacité des adaptations par contrat et négociation. L'objectif de ces expérimentations est d'évaluer le coût de l'utilisation des mécanismes de négociation en termes de *temps d'exécution* des négociations atomiques et de *consommation mémoire*. Pour avoir des premières mesures du surcoût des négociations sans trop d'interférences avec le système applicatif lui-même, ces mesures ne sont pas effectuées dans une application complète mais dans un environnement expérimental simplifié. Ainsi, nous travaillons sur une version légèrement modifiée de l'exemple HelloWorld fourni dans la distribution *Julia* de *Fractal*. Cet exemple définit un composite avec ses deux sous-composants. Les méthodes implémentées par ces derniers ne sont pas vides mais ne font rien de très particulier. Les composants racine et client effectuent essentiellement de la délégation des opérations vers le composant serveur qui lui implémente des opérations simples (calcul arithmétiques, affichage sur la sortie standard...).

A cela, des spécifications externes sont définies sur l'entrée des méthodes, et expriment des contraintes qui prennent en compte divers paramètres (attribut des composants, paramètres d'appels des méthodes...).

Nous rappelons les principales phases du déroulement de la négociation :

1. détection de la violation ;
2. construction et instanciation composants nécessaires à la négociation atomique ;
3. initialisation du composant de négociation ;
4. exécution du composant de négociation ;
   a. consultation de la négociabilité
   b. consultation des propositions
   c. consultation du retrait

Les expérimentations sont effectuées sous les scénariis de négociation suivants :

1. négociation atomique non négociable (phase 2 à 4.a incluses)
2. négociation atomique avec une proposition revalidante [13] (phase 2 à 4.b incluses).
3. négociation atomique avec propositions sans succès et retrait complet [14] (phase 2 à 4.c incluses)

Les mesures sont réalisées en utilisant la bibliothèque de mesures de performance *JAMon* (*Java Application Monitor* en anglais) [JAMon Project (Web Site) 2003]. Cette bibliothèque très légère permet, de façon simple et avec très peu d'impact sur les performances, de collecter diverses données (appels, exceptions, informations temporelles,...) dans différents types d'applications *Java* (applications classiques, EJB, JDBC,...). Enfin, les expérimentations sont faites sur une seul PC (Processeur Intel Pentium M 1.70 Ghz, 1.00 Go RAM) avec l'environnement logiciel suivant : Windows XP Pro., machine virtuelle Java Sun JSDK 1.5.0_10, API Fractal 2.0.2, implémentation Julia 2.5.1.

Les composants applicatifs sont instanciés une seule fois. L'appel de méthode qui déclenche la violation est appelée 10.000 fois, et l'opération est répétée 4 fois. Les mesures de temps dans les tables qui suivent donnent les moyennes sur ces 4 opérations.

9.3.2 Résultats

Temps d'exécution

La table 9.3 présente les temps d'exécution mesurés entre l'appel de méthode et son retour. Le premier cas est réalisé sans contrats (démarrage *Julia*, pas de chargement du système de contrats). Le second cas est effectué avec contrats mais sans négociation. Les violations sont détectées et directement rattrapées dans le code sans négociation pour permettre d'itérer les appels.

	Temps moyen (ms)	**Temps minimum (ms)**	**Temps maximum (ms)**
Sans contrat	9.57	0.0	758.5
Avec contrat sans négociation	14.50	0.0	911

TABLE *9.3 – Temps d'exécution des appels d'une méthode (10.000 appels x 4)*

La table 9.4 présente les temps d'exécution des trois scénarii de négociation atomique présentés précédemment. Le temps mesuré concerne la durée de la négociation atomique uniquement (le *timer* est démarré juste avant la construction de la négociation atomique, et arrêté dès l'obtention du résultat de la négociation qu'il soit en échec ou en réussite. Le temps minimum nul s'explique par le fait que, dans certains cas, le temps d'exécution est en deça de la période d'échantillonnage de l'outil de mesure.

13. La proposition revalidante est placée en milieu d'une liste d'alternatives à 5 éléments.
14. La liste de proposition est ici aussi de taille 5.

	Temps moyen (ms)	Temps minimum (ms)	Temps maximum (ms)
Négociation atomique non négociable	33.187	0.0	761
Négociation atomique avec une proposition revalidante	38.551	0.0	458.25
Négociation atomique avec proposition sans succès et avec retrait complet	37.6295	0.0	1061

TABLE *9.4 – Temps d'exécution d'une négociation atomique (10.000 appels x 4)*

Pour indication, nous avons aussi mesuré les temps moyens complets d'appel et retour de la méthode dans ces trois scénarii de négociation, et ceux-ci sont respectivement de 49.52 ms, 59.03 ms et 52.93 ms

Entre les trois scénarii, les temps d'exécution sont relativement similaires de l'ordre de 36 ms, et il n'y a pas de surcoût notable entre, d'une part, le scénario 1, et d'autre part, les scénarii 2 et 3 de négociation, bien que ces deux derniers fassent intervenir des étapes supplémentaires de consultations et des itérations dans les listes de propositions. La taille des listes a été fixée à 5 (ordre de grandeur < 10) car nous pensons qu'il s'agit d'un ordre de grandeur raisonnable pour des listes devant être écrites.

La comparaison entre les résultats de la table 9.4 et ceux de la table 9.3 montre que les temps d'exécution de la négociation sont du même ordre de grandeur que ceux avec ou sans contrats, à un facteur 3 près. Par ailleurs, le fait que les trois temps d'exécution liés aux négociations soient similaires permet de formuler l'hypothèse selon laquelle ces temps d'exécution proviennent surtout de la phase de construction des composants de négociation (instanciation+initialisation) puisque c'est cette partie qui leur est commune. Il n'y a donc pas, à l'exécution, de forte pénalité induite par les phases de consultation à proprement dites lors de la négociation (scénarii 2 et 3) alors que celles-ci représentent le cœur des interactions. Nous avons effectué des mesures complémentaires qui isolent ce temps de construction et d'initialisation des négociations atomiques "componentisée", et elles montrent que le temps global de la séquence instanciation-démarrage-initialisation-exécution se réparti dans les proportions suivantes : 50% pour l'instantiation, 46% pour l'initialisation, et le reste pour les opérations de démarrage et d'exécution de la négociation (celle-ci conduisant à une clause non négociable dans les mesures effectuées).

Enfin, il est à noter que l'implémentation de la négociation évaluée ici est celle basée composants et décrite dans le chapitre 8. Tout l'environnement nécessaire à la négociation est recrée à chaque fois (instanciation des composants de façon programmatique, chargement des stratégies...), et il n'y a aucun mécanisme d'optimisation mis en œuvre à ce stade. Par ailleurs, il n'y a pas non plus d'optimisation des contrôleurs et des intercepteurs tant pour les composants applicatifs que pour ceux de négociation (contrôleur de cycle de vie présent et absence de fusion de classes).

Consommation mémoire

Les figures qui suivent présentent la mémoire utilisée dans le tas (*heap*) de la machine virtuelle, à partir de laquelle les objets sont alloués, et récupérés par le *garbage collector*. La mémoire est mesurée avec les classes MemoryMXBean et MemoryUsage introduits dans la JDK 1.5. Les mesures sont faites sur deux cas, *i)* avec contrats sans négociation, et *ii)* avec contrats et négociations atomiques.

10.000 appels de méthodes sont lancés, chaque appel déclenche une violation. Dans le premier cas, les violations ne sont pas gérées alors que dans le second cas, pour chaque violation, le scénario de négociation le plus long conduisant au relâchement total est réalisé (phase 2 à 4.c). La consommation mémoire est relevée à intervalle de temps constant (1 unité = 3ms, ce qui représente environ 1/10 de la durée moyenne des négociations atomiques) et donne lieu à la courbe principale (en rouge).

Les figures 9.1 et 9.2 montrent les consommations mémoire avec contrat et sans négociation. Celle-ci oscille autour d'une valeur moyenne de 4.3Mo (cf. Fig 9.1). La figure 9.2 confirme la valeur asymptotique de la consommation mémoire aux alentours de 4.3Mo, par observation sur toute la durée de l'expérience.

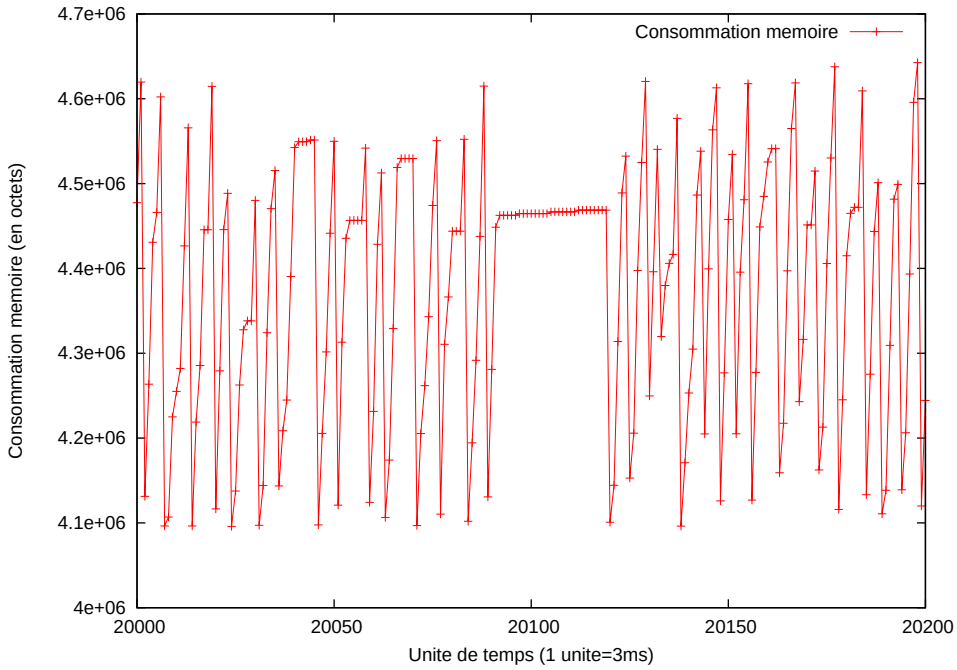


FIGURE *9.1 – Consommation mémoire de l'application, avec contrats sans négociation (intervalle [2000 :20200])*

Les trois figures 9.3, 9.4 et 9.5 suivantes fournissent les consommations détaillées des négociations sur différentes périodes de temps. La courbe principale (toujours en rouge) montre la consommation mémoire de l'application. Les figures 9.3 et 9.4 montrent en plus, les astérisques (en bleu) qui représentent un pointage (temps, consommation mémoire) soit en début, soit en fin de négociation : les astérisques se lisent par couples correspondants à une phase début-fin de négociation. Enfin, les plateaux (en vert) décrivent les périodes de négociations actives et inactives (10^6 pour une négociation active, 0 pour une négociation inactive).

Les figures 9.3 et 9.4 montrent que la consommation mémoire oscille entre 3Mo et 6Mo, et on constate que les pointages (temps, mémoire) de début ou fin de négociation évoluent globalement en fonction de celle de la consommation générale (on les retrouve au niveau des minima et maxima de consommation). Sur la figure 9.4, Nous attribuons la libération de mémoire à t=20500 unité de temps à l'effet du ramasse-miettes. On retrouve ces oscillations dûes au travail du ramasse-miettes sur la figure 9.5.

De plus, la comparaison des pointages de début-fin de négociation montre que dans la grande majorité des cas, la consommation mémoire est plus élevée en fin de négociation qu'en début. Enfin, la figure 9.5 montre l'évolution de ces consommations sur toute la durée de l'expérience, et la comparaison de cette figure avec la figure 9.2 permet de quantifier le surcoût en mémoire de la négociation à environ 0.7Mo (4.3Mo vs 5Mo) pour cette application précise.

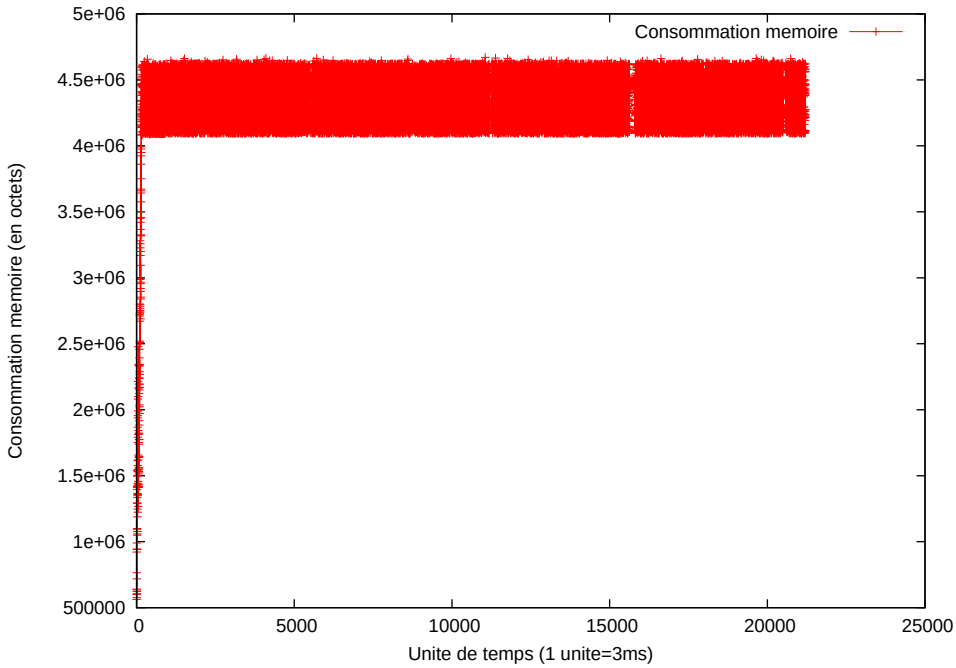


FIGURE 9.2 – *Comportement sur toute la durée de l'expérience*

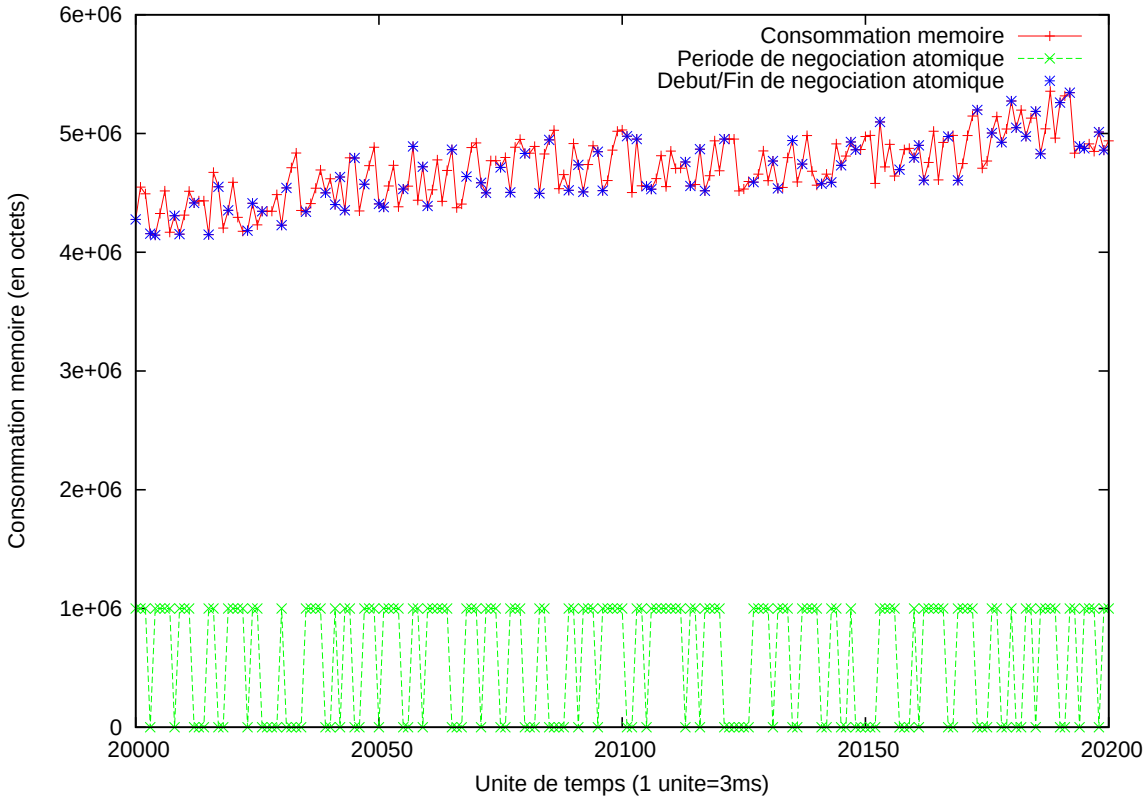


FIGURE 9.3 – *Consommation mémoire de l'application, avec contrats et négociation (intervalle [20000 :20200])*

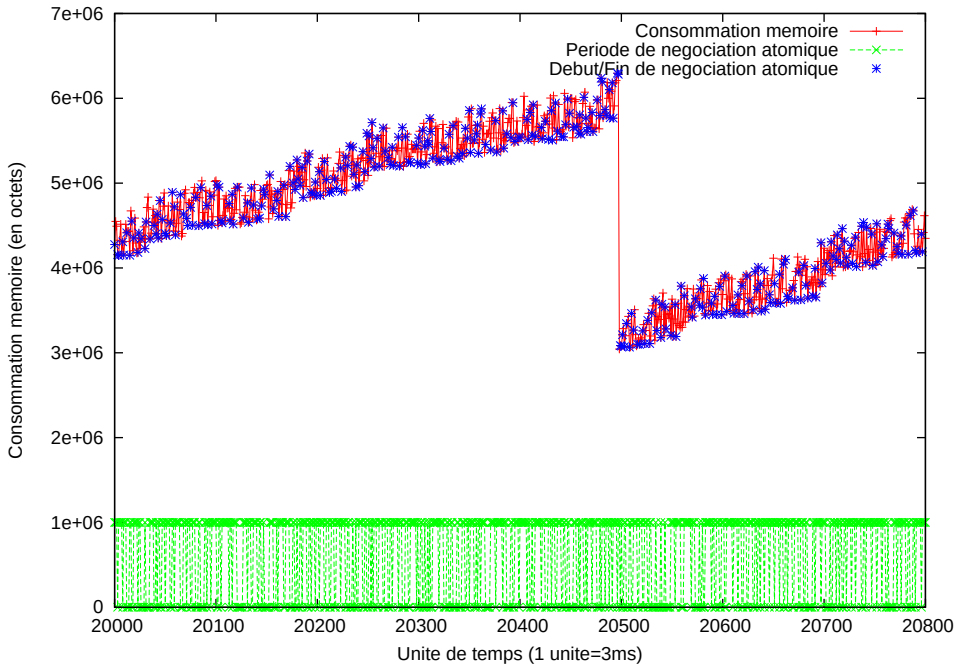


FIGURE 9.4 – *Comportement sur l'intervalle de temps [20000 :20800]*

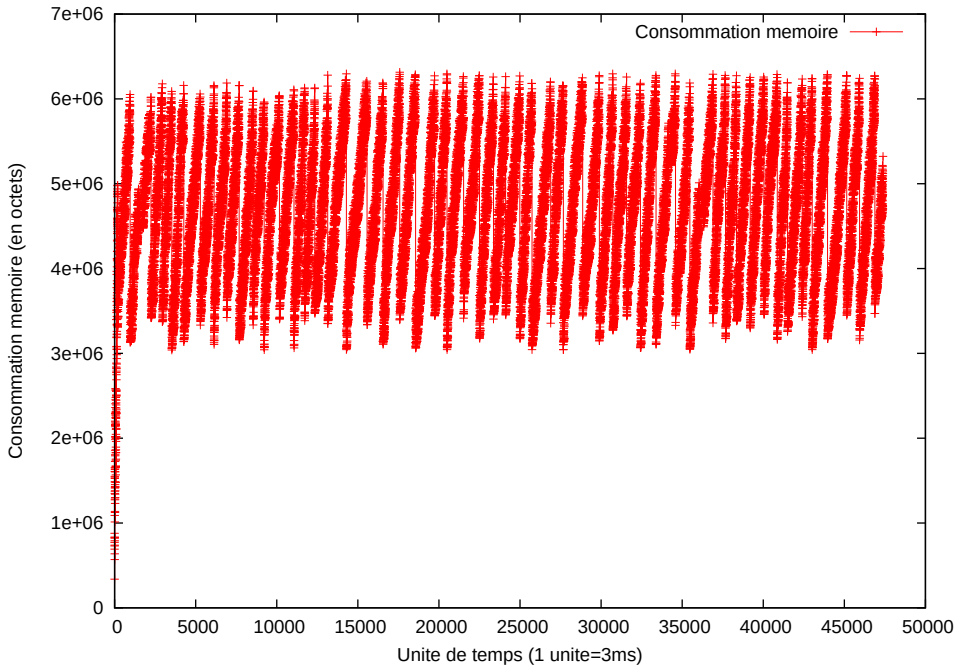


FIGURE 9.5 – *Comportement sur toute la durée de l'expérience*

9.4 Synthèse

L'analyse du modèle de négociation met en évidence plusieurs caractéristiques. Les éléments de base du modèle de négociation respectent l'atomicité des contrats, et l'encapsulation des composants. L'introduction du protocole et des politiques de négociation organisent, de façon simple et orientée, les interactions entre les différentes parties négociantes. Le modèle de négociation définit des principes généraux, et s'intègre en premier lieu dans les plates-formes *Fractal*/*ConFract*. Il conserve et exploite leurs principes généraux (niveaux d'encapsulation des composants, organisation hiérarchique, découpage des contrats, responsabilités...). Une étude plus poussée reste toutefois nécessaire pour évaluer complètement l'extensibilité du modèle de négociation et son application à d'autres plates-formes. Le processus de négociation, quant à lui, met notamment en avant des qualités de simplicité, d'encapsulation des raisonnements des parties impliquées, et d'ouverture. En revanche, en privilégiant cette dernière, les possibilités de contrôle des déroulements des négociations ainsi que des actions effectuées sont (encore plus) limitées (impacts cachés des actions, propagations successives, renégociations en cascade,...). Pour détecter cela, les contrats seront utilisées, et différents paramétrages du processus de négociation pourront être faits pour confiner le déroulement des négociations atomiques autant que possible. D'un point de vue méthodologique, le schéma général qui décrit l'intégration des contrats et de la négociation est présenté. De plus, les trois différents niveaux de variabilité du modèle de négociation (capacités, protocole, politiques) sont identifiés et leurs caractéristiques majeures sont décrites. Ces éléments constituent les points d'entrée pour piloter le système de négociation. Les premières évaluations expérimentales des mécanismes de négociation ont mesuré les temps d'exécution des négociations ainsi que leur consommation mémoire. Ces expérimentations sont réalisées sur une application très simple et elles montrent que les temps d'exécution des négociations ($\simeq$36 ms) sont du même ordre de grandeur, à un facteur 3 près, que les temps d'exécution sans contrat ($\simeq$9 ms) et, avec contrat et sans négociation ($\simeq$14 ms). De plus, les consommations mémoires induisent un surcoût d'environ 0.7Mo pour cette application bien précise. Des mesures de performances complémentaires et plus étendues devront être menées et d'autres expérimentations devront aussi fournir des éléments de mesure de l'efficacité de l'adaptation par un système de contrats négociables sur des systèmes applications de taille et complexité plus importantes.

10
Conclusion et perspectives

Ce mémoire de thèse a présenté un modèle de négociation de contrats pour composants logiciels hiérarchiques. Ce chapitre effectue un rappel des principaux éléments du contexte de ce travail, et conclut ce mémoire en présentant un résumé des contributions et les perspectives de ces travaux.

Les systèmes logiciels modernes sont caractérisés par une complexité croissante ainsi que par l'importance sans cesse accordée aux aspects extrafonctionnels. Face à ces enjeux, le génie logiciel par composants et l'approche contractuelle constituent notamment des solutions pertinentes qui permettent de faciliter le développement par réutilisation d'unités fonctionnelles et d'augmenter la fiabilité des systèmes logiciels. En outre, les exigences fortes en termes continuité et de haute disponibilité des services offerts par les systèmes imposent aussi de prendre en compte leur aspects extrafonctionnels et de proposer continuellement de nouvelles techniques qui permettent aux systèmes logiciels de s'auto-adapter dynamiquement de sorte qu'ils puissent réagir à divers changements et maintenir des qualités satisfaisantes.

Dans ce cadre, ce mémoire présente un travail de thèse autour de l'auto-adaptation dynamique des systèmes à composants logiciels hiérarchiques contractualisés. Il a pour objectif de présenter une vision d'ensemble de l'état de l'art autour des notions de composants contractualisés, d'aspects extrafonctionnels et de négociation dynamique, et de contribuer aux recherches actuelles en proposant une réponse originale qui se base sur ces trois notions. Ce travail présente ainsi un modèle de négociation de contrats qui permet de conduire l'auto-adaptation des systèmes construits par assemblages de composants logiciels hiérarchiques, par le biais de mécanismes de négociation dynamique de contrats intégrés aux composants. Le modèle proposé a pour finalité pratique de fournir des moyens aux administrateurs de grands systèmes logiciels d'automatiser certaines tâches d'administration et de maintenance qui apparaissent lors de la survenue de certaines erreurs du système lui-même ou de son environnement, et à plus long terme, il introduit une nouvelle approche pour la construction de systèmes logiciels autonomes. Les applications concernées sont les grands systèmes logiciels construits sur la base d'une technologie à composants logiciels hiérarchiques et contractualisés. Dans de tels systèmes architecturés par des composants logiciels, les usages des composants et leurs interactions sont spécifiés et surveillés par différents contrats logiciels qui sont construits et évoluent dynamiquement avec le système. Ces contrats observent ainsi le fonctionnement du système et effectuent la détection d'erreurs liées au fonctionnement du système ou à l'environnement. Lorsque des erreurs sont détectées, les contrats sont violés et les mécanismes de négociation de contrats proposés dans ce travail prennent alors en charge l'auto-adaptation.

10.1 Résumé des contributions

Nous avons répondu à la problématique soulevée dans l'introduction en proposant un modèle de négociation de contrats pour composants logiciels hiérarchiques. Ce modèle de négociation définit des processus de négociation qui permettent aux composants de piloter l'adaptation dynamique.

De façon plus précise, notre démarche se base sur les hypothèses suivantes :

- nous considérons les systèmes construits par assemblages hiérarchiques de composants logiciels. Plus spécifiquement, le modèle de composants *Fractal* est utilisé comme technologie de composants de référence, compte tenu de ses qualités de généralité et d'extensibilité ;
- nous nous reposons sur une approche contractuelle pour la spécification et la vérification de diverses propriétés des systèmes à composants logiciels. En particulier, le système de contrats *ConFract* pour composants logiciels hiérarchiques *Fractal* est exploité et les travaux menés dans cette thèse se projettent directement dans ce système ;
- nous proposons de considérer l'auto-adaptation dynamique de façon plus générale comme étant un processus de négociation mené par les composants constituants du système. Contrairement à la majorité des approches actuelles d'auto-adaptation, nous avons soutenu la thèse selon laquelle, il était nécessaire de construire des mécanismes d'auto-adaptation dans une approche montante (*bottom-up* an anglais), en conférant des capacités individuelles aux composants et en les faisant interagir pour qu'ils organisent eux-mêmes les adaptations de leur système.

Les paragraphes suivants détaillent les principales contributions de ce travail.

Modèle de négociation de contrats

Nous avons proposé, dans le chapitre 6, un modèle de négociation qui présente les spécificités suivantes :

- les éléments fondamentaux du modèle de négociation sont définis : les objets de négociation, les parties négociantes, le protocole et les politiques de négociation. Le modèle s'organise autour des négociations atomiques. Chaque négociation atomique porte sur une clause violée d'un contrat, et fait interagir les composants selon un protocole d'interaction simple mais qui définit un initiateur – le gestionnaire de contrats en charge du contrat correspondant – qui contrôle le processus et interagit avec les composants concernés, sur la base de leur responsabilité dans la clause ;
- l'exploitation de ces responsabilités fournit alors un mécanisme très précis pour orienter l'auto-adaptation, et dans cet objectif, deux politiques de négociation s'appuyant sur ces responsabilités fournissent différents déroulements de négociation possibles ;
- les deux politiques se distinguent par le type de responsabilité exploité, la nature des adaptations proposées et le degré de propagation dans la hiérarchie des composants. La politique par concession fait intervenir les composants bénéficiaires d'un contrat et vise à effectuer des adaptations sur les contrats sans possibilité de propagation, alors que la politique par effort fait intervenir le composant garant et vise à réaliser des actions d'adaptation sur les composants, en autorisant d'éventuelles propagations des négociations. Pour effectuer ces propagations et développer complètement la politique par effort, nous avons proposé une contribution auxiliaire détaillée ci-après.

Patrons d'intégration et raisonnement compositionnel

Nous avons aussi proposé, dans le chapitre 7, des patrons d'architecture qui permettent de raisonner compositionnellement sur des propriétés extrafonctionnelles dans la hiérarchie des composants. Ces patrons sont clairement établis à partir d'une analyse et d'une classification des propriétés extrafonctionnelles. Ils permettent d'intégrer et de rendre explicite certaines classes de propriétés extrafonctionnelles

dans les composants. De plus, nous avons défini des entités dédiées pour permettre de raisonner à l'exécution sur les décompositions structurelles des propriétés compositionnelles des composites, compte tenu des composants de leur hiérarchie. Les patrons d'intégration et le support compositionnel sont effectivement exploités pour propager les processus de négociation dans le cadre de la politique par effort, mais ils restent potentiellement réutilisables dans d'autres contextes. Les propositions ont ainsi été effectuées en s'attachant à les abstraire des technologies à composants, néanmoins, la mise en œuvre effective, dans la plate-forme *Fractal*, des patrons d'intégration et du support de raisonnement compositionnel est décrite. Celle-ci est réalisée en réutilisant directement des éléments de base des composants de la plate-forme, et en définissant une nouvelle interface de contrôle qui permet de gérer les diverses propriétés compositionnelles des composants composites paramétrés. Les différents éléments du modèle proposé sont illustrés et discutés au travers de plusieurs exemples de contrats et de scénarii de négociation extraits du cas d'étude présenté dans l'annexe A. Cette annexe présente le cas d'étude de façon plus complète et fournit des exemples de négociation complémentaires.

Intégration, évaluation et expérimentations

Ces contributions s'intègrent dans la plate-forme de composants *Fractal*. Une implémentation du modèle de négociation est réalisée en se basant sur la première version du système *ConFract* et sur la dernière version 2.5 de l'implémentation de référence *Julia* de la plate-forme *Fractal*. Le chapitre 8 fournit une vision complète de l'architecture du système de négociation et des interactions entre les principaux éléments, implémentés eux-mêmes sur la base de composants *Fractal* contenus dans les membranes des composants métiers concernés. En particulier, l'implémentation du système de négociation présente les caractéristiques suivantes. Elle respecte les règles de visibilité des composants, réutilise avantageusement la notion de partage de composants et exploite une caractéristique majeure, proposée dans la version 2.5 de *Julia*, d'architecturer les membranes de composants par des composants. Différents composants sont définis dans ce sens, ils sont créés à l'exécution lors des déclenchements des processus de négociation et sont associés aux composants métiers concernés par les négociations, par reconfiguration dynamique de leur membrane. Enfin, le chapitre 9 fournit des éléments d'évaluation qualitative à la fois du modèle et des processus de négociation. Il présente des éléments méthodologiques pour l'utilisation du modèle de négociation, et analyse les premières évaluations de performances, en termes de temps d'exécution et de consommation mémoire, réalisées dans le cadre d'une application très simple.

10.2 Perspectives

Sur la base du travail réalisé durant cette thèse, nous identifions les perspectives de recherche suivantes. Les perspectives directes à court et moyen terme sont d'abord présentées avant de discuter les axes de recherche qui sont liés à des travaux à plus long terme.

Perspectives à court terme

Stabilité et cohérence du système de négociation Comme nous l'avons évoqué dans le chapitre 9, une des principales limites des processus de négociation concerne le contrôle général des négociations (réalisation d'actions d'adaptations incontrôlées, propagations en cascade, cycle de violations et re-négociations de contrats dépendants, etc.). Il est possible d'appliquer des solutions locales pour traiter individuellement chacune de ses limites (désactivation des contrats violés en boucle, définition d'un nombre maximal de niveaux de propagation, définition d'une durée maximale de négociation,...). Toutefois, dans une approche plus complète, une piste intéressante de recherche consisterait plutôt à étudier dans quelle mesure il est possible d'exprimer ou d'inférer les relations et dépendances qui peuvent

exister entre les éléments des négociations atomiques (clauses dépendantes, propriétés liées, etc.). Cela pourrait se faire, par exemple, par observation du système, des contrats et des négociations à l'exécution, par analyse statique des clauses des contrats, ou encore par analyse des dépendances entre propriétés extrafonctionnelles contractualisées. Ainsi, comme ces limites découlent surtout des relations d'interdépendances qui peuvent exister, la meilleure compréhension de ces dépendances permettrait de les gérer d'une façon plus globale.

Modélisation d'autres propriétés extrafonctionnelles Lors de l'intégration des propriétés extrafonctionnelles dans les composants, notre étude était restreinte aux propriétés quantitative, de bas-niveau et orthogonales aux composants. Il serait bien évidemment intéressant de prolonger cette étude en considérant d'autres propriétés extrafonctionnelles, notamment des propriétés temporelles. Cette étude pourrait ainsi s'appuyer sur des systèmes d'interception et de surveillance à l'exécution, des approches basées sur des scénarii de simulation [Bondarev *et al.* 2004] ou encore des canevas d'analyse statique [Defour *et al.* 2004a]. De plus, comme nous l'avons souligné, une dimension de plus en plus importante réside dans les possibilités de pouvoir *raisonner compositionnellement* sur des propriétés extrafonctionnelles d'un système, compte tenu des entités d'architectures qui la structurent, tout cela dans le but final de faciliter l'évaluation des qualités du système global. Un autre prolongement de ce travail pourrait ainsi consister à étudier d'autres formes de *composabilité* des propriétés extrafonctionnelles, qui prendrait par exemple en compte d'autres types d'opérateurs ou de modèles de composition (automates, modèles markoviens...) [Reussner *et al.* 2003], ou des éléments ou des propriétés qui ne découlent pas uniquement de composants [Crnkovic *et al.* 2004] (profils d'utilisation, influences de l'environnement...).

Expérimentations et évaluation de performances complémentaires Les premières évaluations quantitatives se placent dans le cadre d'une application très simple et mesurent le coût des négociations en termes de temps d'exécution et de consommation mémoire. Les surcoûts mesurés découlent directement du choix d'implémenter le système de négociation sous la forme de composants. Par ailleurs, celle-ci est aussi réalisée sans optimisation, ni sur *Julia*, ni sur le système lui-même (instanciation des composants, chargement des stratégies, etc.). Une perspective directe consisterait donc à travailler sur l'optimisation de certains éléments de l'implémentation, et nous pensons que l'architecture à composants du système de négociation peut à l'inverse être utilisée pour déjà identifier et isoler certains points d'optimisation (définition de composants singletons pour les fabriques ou les builders, réutilisation de composants de stratégies, etc.). De plus, selon les contraintes des applications cibles, il sera toujours possible d'effectuer des optimisations similaires au niveau de l'implémentation *Julia* (fusion des objets d'interception et de contrôle des composants), et un retour à une version tout objet reste envisageable si les contraintes sont fortes. De nouvelles expérimentations sur le JDK 1.6 sont aussi à effectuer. Du point de vue de l'efficacité de l'adaptation, puisqu'aucune mesure relative à cela n'est effectuée à ce stade, nous comptons mettre en place une évaluation plus complète des mécanismes de négociation sur des applications de taille et de complexité significatives, à commencer par celle qui fait l'objet de notre étude de cas. Cette évaluation devra notamment définir les métriques de réussite de l'adaptation (nombre d'appels réussis, minimisation des montées en charge des ressources...) ainsi que l'environnement de simulation (injections de charges diverses).

Application du modèle à d'autres plates-formes Le modèle de négociation proposé est actuellement validé sur la plate-forme de composants logiciels *Fractal*. Dans l'optique de généraliser le modèle de négociation, il serait pertinent d'étudier l'application du modèle à d'autres types d'architectures et de plates-formes. En particulier, les architectures orientées service [Erl 2005] et les plates-formes de web services [Weerawarana *et al.* 2005] constituent un domaine d'étude fort intéressant, car elles aussi in-

tègrent et proposent de plus en plus des formes de contrats et de négociation. Ainsi, des travaux de recherche à court terme consisteraient i) à étudier l'utilisation des mécanismes de négociation en tant que tels pour l'établissement de contrats/accords qui vont régir les interactions fonctionnelles fournisseurs et consommateurs de services, ii) à exploiter les mécanismes de négociation en tant que moyens d'adaptation pour rétablir des contraintes extrafonctionnelles non satisfaites au niveau des interactions métiers (*Service Level Objectives* en anglais), et iii) à exploiter les mécanismes de négociation pour concevoir des applications orientées services auto-adaptatives dès lors qu'ils sont implémentées sous la forme de composants. Il est à noter que compte tenu de la généralité du modèle *Fractal*, une première étape de ces travaux serait de *virtualiser* une interaction orientée service – consommateur/fournisseur – par des composants *Fractal* et de travailler directement sur ces derniers. Dans cette démarche, une première boîte à outils a déjà été réalisée pour établir plus facilement des ponts entre composants logiciels et web services [Collet *et al.* 2007a]. Ces derniers automatisent l'exposition d'interfaces de composants sous la forme de web services, ainsi que l'intégration d'un web service externe dans un assemblage de composants. La représentation architecturale, par des composants, d'*orchestrations de web services* est aussi à l'étude. Enfin, une perspective à beaucoup long terme est de proposer une *virtualisation* de systèmes logiciels dans le but d'intégrer et de réaliser, de façon homogène, diverses fonctionnalités d'administration et de maintenance. Compte tenu des qualités du modèle *Fractal*, une telle virtualisation pourrait être implémentée sur ce modèle et celle-ci offrirait alors une abstraction et une vision hiérarchique des systèmes dans laquelle *i)* les composants implémentent des fonctionnalités propres ou interfaçent des systèmes patrimoniaux existants, et *ii)* les différentes opérations d'administration et de maintenance sont réalisées en prenant appui sur les fonctionnalités de contrôle (réalisation d'interceptions, intégration de divers services de contrôle,...).

Axes de recherches

Suivi des évolutions du système, des contrats et des adaptations Les contrats du système *ConFract*, sur lequel s'appuie principalement nos propositions, suivent le cycle de vie des composants et sont mis à jour lors de leurs reconfigurations dynamiques. Toutefois, il n'y a pas réellement de maintien des évolutions successives des contrats. Pour augmenter les possibilités d'analyse de ces éléments (composants, contrats, adaptations) et de leurs relations mutuelles, il serait intéressant de capturer et de maintenir à l'exécution, d'une part les évolutions successives des contrats déjà réifiés, mais aussi celles qui concernent l'architecture des composants et les actions d'adaptations effectuées par la négociation. Par exemple, cela pourrait se faire par une architecture à couches dans laquelle : i) le niveau de base représenterait les instances des composants applicatifs courants, ii) un second niveau maintiendrait les évolutions de l'architecture des composants, initialisée avec l'architecture initiale et mis à jour selon les reconfigurations dynamiques des composants, iii) un troisième niveau maintiendrait les évolutions des contrats, initialisé par les contrats issus des spécifications initiales et mis à jour selon les reconfigurations des composants, et iv) le quatrième niveau maintiendrait les actions d'adaptations résultantes de la négociation, effectuées tant sur le niveau des contrats que sur le niveau des composants. Ainsi, il serait possible de conserver les historiques des évolutions de chacun de ces éléments mais aussi d'envisager des retours à des configurations 'composants-contrats-adaptations' stables.

Pilotage global de l'auto-adaptation Les mécanismes d'auto-adaptation reposent sur les négociations atomiques. Celles-ci sont définies dans un contexte (portée spatiale, composants impliqués, clause concernée) bien précis, et les négociations atomiques peuvent donc se dérouler de façon indépendante les unes des autres. Dans une telle approche décentralisée, il n'est actuellement pas possible d'avoir une vue globale des différentes négociations atomiques qui se déroulent et cela empêche donc définir des *stratégies* globales de contrôle de ces différentes négociations atomiques. Dans cette optique, une

autre piste intéressante de recherche consisterait à étudier dans quelle mesure il est possible de spécifier un système de négociation, ou plus généralement d'auto-adaptation, de façon à ce qu'il puisse piloter et optimiser l'adaptation du système global à partir d'objectifs de haut-niveau et en agissant sur les mécanismes d'adaptation sous-jacents – les négociations atomiques dans notre cadre. Cela pourrait, par exemple, consister à configurer les mécanismes d'auto-adaptation, à gérer des conflits entre actions d'adaptations locales, ou encore à piloter de négociations dépendantes. Toutefois, pour mettre en œuvre cela, des études plus approfondies sont bien évidemment requises.

Méthodologie pour l'adaptation Dans ce travail de thèse, comme dans la plupart des techniques d'adaptation, les mécanismes mis en œuvre sont automatisés à condition que les logiques d'adaptation soient fournies. En conséquence, le point crucial est à ce niveau et compte tenu de l'importance croissante de l'auto-adaptation et de l'émergence de très nombreux mécanismes pour supporter cela, il devient maintenant pertinent de s'attacher à développer une certaine méthodologie de l'adaptation. En particulier, cette méthodologie pourra adresser les questions suivantes : *i)* étant donné un système logiciel donné qui fonctionne dans un environnement d'exécution, comment identifier les événements sur lesquels réagir, puis les *bonnes* actions d'adaptations à mettre en œuvre en réponse à ces erreurs ?, *ii)* comme un des buts de l'auto-adaptation est de rétablir un certain niveau de fonctionnement *acceptable* des systèmes, dans quelle mesure est-il possible d'évaluer quantitativement, peut-être par la définition de critères d'optimalité ou de sous-optimalité sur des propriétés globales du système, que les adaptations marchent effectivement ? *iii)* pour des systèmes qui fonctionnent sur des durées réellement longues, dans quelle mesure est-il possible de ré-évaluer l'adéquation des logiques d'adaptation initiales et de les faire évoluer au cours du temps ?

⋆

10.3 Publications liées à cette thèse

Les travaux réalisés dans cette thèse ont été publiés à ce jour dans les articles et rapports de recherche suivants :

- **Article de revue internationale :**
  - *Compositional Patterns of Non-Functional Properties for Contract Negotiation*
    Auteurs : Hervé Chang et Philippe Collet
    Journal : Journal of Software (JSW), Academy publishers, vol. 2(2), pages 52–63, Août 2007
- **Article de revue nationale :**
  - *Négociation de contrats, des systèmes multi-agents aux composants logiciels*
    Auteurs : Hervé Chang et Philippe Collet
    Journal : L'Objet Hermès, Numéro spécial "Composants et Systèmes Multi-Agents", vol. 12(4), pages 73–102, Décembre 2006
- **Actes de conférences internationales avec comité de lecture :**
  - *Patterns for Integrating and Exploiting Some Non-Functional Properties in Hierarchical Software Components*
    Auteurs : Hervé Chang et Philippe Collet
    Conférence : Proceedings of 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (IEEE ECBS'07), pages 83–92, Mars 2007. IEEE Computer Society
  - *From Components to Autonomic Elements Using Negotiable Contracts*
    Auteurs : Hervé Chang, Philippe Collet, Alain Ozanne, et Nicolas Rivierre
    Conférence : Proceedings of 3rd International Conference on Autonomic and Trusted Computing (ATC'06), volume 4158, pages 78–89, Septembre 2006. Springer-Verlag LNCS
  - *Fine-grained Contract Negotiation for Hierarchical Software Components*
    Auteurs : Hervé Chang et Philippe Collet
    Conférence : Proceedings of 31st EUROMICRO Conference on Software Engineering and Advanced Applications, CBSE Track (Euromicro-SEAA'05), pages 28–35, Septembre 2005. IEEE Computer Society
- **Actes de conférences nationales avec comité de lecture :**
  - *Eléments d'architecture pour la négociation de contrats extrafonctionnels*
    Auteurs : Hervé Chang et Philippe Collet
    Conférence : Proceedings of 1ère Conférence francophone sur les Architectures Logicielles (CAL'06), L'Objet, pages 151–167, Septembre 2006. Hermès
  - *Vers la négociation de contrats dans les composants logiciels hiérarchiques*
    Auteurs : Hervé Chang et Philippe Collet
    Conférence : Proceedings of 11e Conférence Langages et Modèles à Objets (LMO'05), vo-

lume 11 of L'Objet, pages 239–252, Mars 2005. Hermès

- **Journées nationales avec comité de lecture :**

  – *Des patrons d'intégration de propriétés extrafonctionnelles dans les composants logiciels hiérarchiques*
  Auteur : Hervé Chang
  Journée : 4e Journées du groupe Objets, Composants et Modèles (OCM'06), GDR ALP, pages 5–12, Mars 2006

  – *Négociation de contrats - des systèmes multiagents aux composants logiciels*
  Auteurs : Hervé Chang et Philippe Collet
  Journée : 1ère Journée Multi-Agent et Composant (JMAC'04), GDR ALP, pages 44–56, Novembre 2004

- **Papiers courts :**

  – *Some Autonomic Features of Hierarchical Components with Negotiable Contracts*
  Auteurs : Hervé Chang, Philippe Collet, Alain Ozanne, et Nicolas Rivierre
  Conférence : Proceedings of 3rd IEEE International Conference on Autonomic Computing (IEEE ICAC'06), Short paper, pages 285–286, June 2006. IEEE Computer Society

- **Rapports de recherche et livrables de contrats :**

  – *Spécification d'intégration du modèle de négociation dans Fractal/ConFract*
  Auteurs : Hervé Chang et Philippe Collet
  Livrable F3.2 France Télécom, Contrat recherche externe n^o 46132097, Laboratoire I3S UNSA/CNRS, 31 pages, Juin 2007

  – *Components and Services : A Marriage of Reason*
  Auteurs : Philippe Collet, Thierry Coupaye, Hervé Chang, Lionel Seinturier et Guillaume Dufrêne
  Rapport Recherche I3S/RR-2007-17-FR Projet Rainbow, 16 pages, Mai 2007.

  – *Définition du modèle de négociation*
  Auteurs : Hervé Chang et Philippe Collet
  Livrable F2.2 France Télécom, Contrat recherche externe n^o 46132097, Laboratoire I3S UNSA/CNRS, 73 pages, Décembre 2006

  – *État de l'art sur la négociation dynamique*
  Auteurs : Hervé Chang et Philippe Collet
  Livrable F1.2 France Télécom, Contrat recherche externe n^o 46132097, Laboratoire I3S UNSA/CNRS, 84 pages, Mars 2006

Bibliographie

J. AAGEDAL : *Quality of Service Support in Development of Distributed Systems*. Thèse de doctorat, University of Oslo, Department of Informatics, Faculty of Mathematics and Natural Sciences, March 2001. ISSN 1501-7710, No. 147. 31

M. ABADI et L. LAMPORT : Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993. ISSN 0164-0925. 16

J. M. ANDREOLI et S. CASTELLANI : Towards a Flexible Middleware Negotiation Facility for Distributed Components. *In DEXA'01 : Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, p. 732, Washington, DC, USA, 2001. IEEE Computer Society. 18

J. H. ANDREWS et Y. ZHANG : Broad-spectrum Studies of Log File Analysis. *In ICSE '00 : Proceedings of the 22nd international conference on Software engineering*, p. 105–114, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-206-9. 20

A. ANDRIEUX, K. CZAJKOWSKI, A. DAN, K. KEAHEY, H. LUDWIG, J. PRUYNE, J. ROFRANO, S. TUECKE et M. XU : Web Services Agreement (WS-Agreement) Specification, GWD-R (Proposed Recommendation) to Global Grid Forum, Version 2005/09. Rap. tech., Global Grid Forum, 2005. Available at : http ://www.ggf.org/. 18

A. ANDRIEUX, A. DAN, K. KEAHEY, H. LUDWIG et J. ROFRANO : Negotiability Constraints in WS-Agreement, Version 0.1. Document to GRAAP-WG Meeting. Rap. tech., Global Grid Forum, 2004. 18

A. ANJOMSHOAA, F. BRISARD, M. DRESCHER, D. FELLOWS, A. LY, S. MCGOUGH, D. PULSIPHER et A. S. (EDS.) : Job Submission Description Language (JSDL) Specification, Version 1.0, GFD-R-P.056. GGF. Rap. tech., Global Grid Forum, 2005. 17

M. ATIGHETCHI, P. P. PAL, C. C. JONES, P. RUBEL, R. E. SCHANTZ, J. P. LOYALL et J. A. ZINKY : Building auto-adaptive distributed applications : The quo-apod experience. *In Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCSW'2003)*, p. 104, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1921-0. 35

C. AURRECOECHEA, A. T. CAMPBELL et L. HAUW : A Survey of QoS Architectures. *Multimedia Systems*, 6(3):138–151, 1998. ISSN 0942-4962. 36

J. AUSTIN : *How to do things with words*. Oxford university press, 1962. 52

J. AUSTIN, M. FLETCHER et T. JACKSON : Distributed Aero-Engine Condition Monitoring and Diagnosis on the GRID : DAME. *In Proceedings of the 17th International Congress and Exhibition on Condition Monitoring and Diagnostic Engineering Management (COMADEM 2004 International)*, 2004. 18

Z. BALOGH, M. LACLAVÍK et L. HLUCHÝ : Model of Negotiation and Decision Support for Goods and Services. *In Proceedings of XXIInd International Colloquium ASIS 2000 - Advanced Simulation of Systems*, Ostrava, Czech Republic, 2000. 56

M. BARBACCI, M. KLEIN, T. LONGSTAFF et C. WEINSTOCK : Quality Attributes. Rap. tech. CMU/SEI-95-TR-021, CMU/SEI, December 1995. 2, 97

C. BARTOLINI, C. PREIST et N. JENNINGS : A software framework for automated negotiation. Rap. tech. HPL-2002-2, HP Labs, 2002. 58

L. BASS, P. CLEMENTS et R. KAZMAN : *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998. ISBN 0-201-19930-0. 2

R. BAUMGARTL, M. BORRISS, H. HÄRTIG, C.-J. HAMANN, M. HOHMUTH, L. REUTHER, S. SCHÖNBERG et J. WOLTER : Dresden Realtime Operating System. *In Proceedings of the Workshop of System-Designed Automation (SDA'98)*, Dresden, Germany, March 1998. 34

C. BECKER et K. GEIHS : MAQS - Management for Adaptive QoS-enabled Services. *In IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, USA, 1997. 30

C. BECKER et K. GEIHS : Generic qos-support for corba. *In ISCC '00 : Proceedings of the Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, p. 60, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0722-0. 30

M. BENYOUCEF, R. KELLER, S. LAMOUROUX, J. ROBERT et V. TRUSSART : Towards a generic e-negotiation platform. *In Proc. of Conference on Re-Technologies for Information Systems, pp. 95-109, Zurich, Switzerland,*, February 2000. 58

M. BERTOA et A. VALLECILLO : Quality Attributes for COTS Components. *In ECOOP'2002 QAOOSE Workshop*. Springer LNCS, 2002. 97

A. BEUGNARD, J.-M. JÉZÉQUEL, N. PLOUZEAU et D. WATKINS : Making components contract aware. *Computer*, 32(7):38–45, 1999. ISSN 0018-9162. 4, 15, 19

A. P. BLACK, J. HUANG, R. KOSTER, J. WALPOLE et C. PU : Infopipes : an abstraction for multimedia streaming. *Multimedia Syst.*, 8(5):406–419, 2002. ISSN 0942-4962. 36

G. S. BLAIR, G. COULSON, A. ANDERSEN, L. BLAIR, M. CLARKE, F. COSTA, H. DURAN-LIMON, T. FITZPATRICK, L. JOHNSTON, R. MOREIRA, N. PARLAVANTZAS et K. SAIKOSKI : The Design and Implementation of Open ORB 2. *IEEE Distributed Systems Online*, 02(6), 2001. ISSN 1541-4922. 3

G. S. BLAIR, G. COULSON, L. BLAIR, H. DURAN-LIMON, P. GRACE, R. MOREIRA et N. PARLAVANTZAS : Reflection, self-awareness and self-healing in openorb. *In Proceedings of the first workshop on Self-healing systems (WOSS'2002)*, p. 9–14, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-609-9. 33, 36

E. BONDAREV, J. MUSKENS, P. de WITH, M. CHAUDRON et J. LUKKIEN : Predicting Real-Time Properties of Component Assemblies : A Scenario-Simulation Approach. *In EUROMICRO-SEAA'2004*. IEEE Computer, 2004. 140

A. BOUCH et M. SASSE : Why Value Is Everything : A User-Centered Approach to Internet Quality of Service and Pricing. *In IWQoS '01 : Proceedings of the 9th International Workshop on Quality of Service*, p. 59–74, London, UK, 2001. Springer-Verlag. ISBN 3-540-42217-X. 36

C. BRAGA et A. SZTAJNBERG : Towards a rewriting semantics for a software architecture description language. *In Proc. of the Brazilian Workshop on Formal Methods*, p. 149–168. Electronic Notes in Theoretical Computer Science, Elsevier, May 2004. 33

G. BRAHNMATH, R. RAJE, A. OLSON, B. BRYANT, M. AUGUSTON et C. BURT : A Quality of Service Catalog for Software Components. *In Proceedings of the Southeastern Software Engineering Conference*, p. 513–520, 2002. 24

J.-P. BRIOT et Y. DEMAZEAU : *Principes et architecture des systèmes multi-agents (Traité IC2, Informatique et systèmes d'information)*. Hermes - Lavoisier, November 2001. 5, 41, 50

A. BROWN : *Large-Scale, Component Based Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. ISBN 013088720X. 1

E. BRUNETON, T. COUPAYE et J.-B. STEFANI : The Fractal Component Model. Specification, Technical Report v1, v2, The ObjectWeb Consortium, 2002. http ://fractal.objectweb.org. 106, 107, 110

E. BRUNETON, T. COUPAYE, M. LECLERCQ, V. QUÉMA et J.-B. STEFANI : An open component model and its support in java. *In Component-Based Software Engineering, 7th International Symposium, CBSE 2004*, vol. 3054 de *Lecture Notes in Computer Science*, p. 7–22. Springer, May 2004. ISBN 3-540-21998-6. 71

E. BRUNETON, T. COUPAYE, M. LECLERCQ, V. QUÉMA et J.-B. STEFANI : The fractal component model and its support in java : Experiences with auto-adaptive and reconfigurable systems. *Softw. Pract. Exper.*, 36(11&dash ;12):1257–1284, 2006. ISSN 0038-0644. 2, 6, 71

M. BUCHI et W. WECK : A Plea for Grey-Box Components. *In Proceedings, Foundations of Component-Based Systems Workshop*, p. 287–308, Zurich, Switzerland, 1997. 14, 19

M. BUCHI et W. WECK. : The Greybox Approach : When Blackbox Specifications Hide Too Much. Rap. tech. 297, Turku Center for Computer Science,, 1999. 14, 19

J. BUISSON : *Adaptation dynamique de programmes et composants parallèles*. Thèse de doctorat, INSA Rennes, Rennes, Septembre 2006. 38

S. BUSSMANN et J. MULLER : A negotiation framework for co-operating agents. *In* D. S. M, éd. : *Proc. of the Special Interest Group on Cooperating Knowledge Based Systems (CKBS-SIG)*, p. 1–17, Dake Centre, University of Keele, 1992. 59

S. J. CAMMARATA, D. MCARTHUR et R. STEEB : Strategies of cooperation in distributed problem solving. *In IJCAI*, p. 767–770, 1983. 44, 46, 50

C. CANAL, L. FUENTES, J. M. TROYA et A. VALLECILLO : Extending CORBA Interfaces with p-Calculus for Protocol Compatibility. *In TOOLS '00 : Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*, p. 208, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0731-X. 19

R. CERQUEIRA, S. ANSALONI, O. LOQUES et A. SZTAJNBERG : Deploying Non-Functional Aspects by Contract. *In Middleware 2003 - 2nd Workshop on Reflective and Adaptive Middleware, Middleware2003 Companion*, p. 90–94, Rio de Janeiro, Brazil, June 2003. 33

S. CHATTERJEE, B. SABATA et M. BROWN : Adaptive qos support for distributed java-based applications. *In Proc. of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Saint-Malo, France, May 02-05 1999. 35

A. CHAVEZ et P. MAES : Kasbah : An Agent Marketplace for Buying and Selling Goods. *In Proc. of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 1996. 55, 56

D. CHEFROUR : *Plate-forme de composants logiciels pour la coordination des adaptations multiples en environnement dynamique*. Thèse de doctorat, Université de Rennes I, Rennes, Novembre 2005. 37

V. CHEVRIER : Coordination et structuration des échanges par négociation dans les systèmes multi-agents. *Actes journées multi-agents du GDR-PRC Intelligence Artificielle Nancy*, Decembre 1992. 51, 56

E. CHRISTENSEN, F. CURBERA, G. MEREDITH et S. WEERAWARANA : Web Services Description Language (WSDL), Version 1.1, W3C. Rap. tech., World Wide Web Consortium, 2006. http ://www.w3.org/TR/wsdl. 17

J. CLAPP et F. STANTEN : A Guide to Total Software Quality Control. Rap. tech., MITRE CORP BEDFORD MA, 1992. 23

P. R. COHEN et H. J. LEVESQUE : Communicative actions for artificial agents. *In ICMAS*, p. 65–72, 1995. 52

P. COLLET, T. COUPAYE, H. CHANG, L. SEINTURIER et G. DUFRÊNE : Components and Services : A Marriage of Reason. Research Report I3S-RR2007-17FR, I3S Lab, Sophia Antipolis, France, May 2007a. 141

P. COLLET, A. OZANNE et N. RIVIERRE : Enforcing Different Contracts in Hierarchical Component-Based Systems. *In Proceedings of 5th International Symposium Software Composition (SC'2006)*, vol. 4089 de *Lecture Notes in Computer Science*, p. 50–65. Springer Verlag, 2006. ISBN 3-540-37657-7. 16

P. COLLET, A. OZANNE et N. RIVIERRE : Towards a Versatile Contract Model to Organize Behavioral Specifications. *In Proceedings of 33rd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'07)*, vol. to appear de *LNCS*, p. 4362, Harrachov, Czech Republic, 2007b. Springer Verlag. 16

P. COLLET, R. ROUSSEAU, T. COUPAYE et N. RIVIERRE : A contracting system for hierarchical components. *In Component-Based Software Engineering, 8th International Symposium, CBSE 2005, Proceedings*, vol. 3489 de *Lecture Notes in Computer Science*, p. 187–202. Springer, May 2005. ISBN 3-540-25877-9. 6, 16, 73, 177

J. COLLINS, S. JAMISON, B. MOBASHER et M. GINI : MAGNET : A Market Architecture for Multi-Agent Contracting. *In Proc. of the International Conference on Autonomous Agents (Agents)*, Minneapolis, MN, USA, May 1998a. 57

J. COLLINS, B. YOUNGDAHL, S. JAMISON, B. MOBASHER et M. GINI : A market architecture for multi-agent contracting. *In Proc. of the Interbational Conference on Autonomous Agents*, p. 285–292, Minneapolis, May 1998b. 57

COMQUAD PROJECT (WEB SITE), 2001 : http ://www-st.inf.tu-dresden.de/comquad/. 34

S. CONRY, R. MEYER et V. LESSER : *Multistage negotiation in distributed planning*, p. 367–384. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-63-X. 54

D. CORKILL : Hierarchical Planning in a Distributed Problem-Solving Environment. *In Proc. 7th Intl. Joint Conf. on AI*, Tokyo, January 1979. 46

G. COULSON, G. BLAIR, M. CLARK et N. PARLAVANTZAS : The design of a configurable and reconfigurable middleware platform. *ACM Distributed Computing Journal*, 15(2):109–126, April 2002. 2, 36

T. COUPAYE, J.-B. STEFANI et G. BLAIR : *(*To Appear*) Software components - The Fractal initiative*. Annals of Telecommunications, Paris, France, 2008. 6

I. CRNKOVIC, M. LARSSON et O. PREISS : Concerning Predictability in Dependable Component-Based Systems : Classification of Quality Attributes. *In DSN'2004 WADS Workshop*. Springer - LNCS, 2004. 27, 140

K. CZAJKOWSKI, A. DAN, J. ROFRANO, S. TUECKE et M. XU : Agreement-based Grid service management (OGSI-Agreement). GRAAP-WG Author Contribution. Rap. tech., Global Grid Forum, 2003. Available at : http ://www.ggf.org/. 18

K. CZAJKOWSKI, I. FOSTER, C. KESSELMAN, V. SANDER et S. TUECKE : SNAP : A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *In Proc. of 8th Workshop on Job Scheduling Strategies for Parallel Processing*, Edinburgh Scotland, July 2002. 18

M. DALMAU, S. LAPLACE et P. ROOSE : Méthode de choix d'une configuration de composants logiciels optimale guidée par l'évaluation de la qualité de service. *In Journées Composants 2004, Association ACM-SIGOPS de France*, Lille, France, Mars 2004. 37

DAN, K. KEAHEY, H. LUDWIG et J. ROFRANO : Guarantee Terms in WS-Agreement, Version 0.1. Document to GRAAP-WG Meeting. Rap. tech., Global Grid Forum, 2004. 18

M. D'ARIENZO, APESCAPÈ, S. ROMANO et G. VENTRE : The service level agreement manager : Control and management of phone channel bandwidth over premium ip networks. *In Proceedings of the 15th International Conference on Computer Communication (ICCC'02)*, p. 421–432, 2002. ISBN 0-7965-2276-9. 18

DARPA KNOWLEDGE SHARING EFFORT (WEB SITE) : http ://www.cs.umbc.edu/kse/. 52

E. M. DASHOFY, A. van der HOEK et R. N. TAYLOR : Towards Architecture-Based Self-Healing Systems. *In Proceedings of the first workshop on Self-healing systems (WOSS'2002)*, p. 21–26, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-609-9. 32

P.-C. DAVID. : *Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation*. Thèse de doctorat, École des Mines de Nantes et Université de Nantes,, Nantes, Juillet 2005. 37

R. DAVIS et R. G. SMITH : *Negotiation as a metaphor for distributed problem solving*, p. 333–356. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-63-X. 45

O. DEFOUR, J.-M. JÉZÉQUEL et N. PLOUZEAU : Applying CLP to Predict Extra-Functional Properties of Component-Based Models. *In Proceedings of International Conference on Logic Programming (ICLP'04)*. Springer - LNCS, 2004a. 140

O. DEFOUR, J.-M. JÉZÉQUEL et N. PLOUZEAU : Extra-functional Contract Support in Components. *In CBSE'7*, LNCS. Springer-Verlag, May 2004b. 30

N. K. DIAKOV, H. J. BATTERAM, H. ZANDBELT et M. van SINDEREN : Design and Implementation of a Framework for Monitoring Distributed Component Interactions. *In IDMS '00 : Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, p. 227–240, London, UK, 2000. Springer-Verlag. ISBN 3-540-41130-5. 20

H. DURAN-LIMON, G. BLAIR et G. COULSON : Adaptive resource management in midlleware : A survey. *IEEE Distributed Systems online*, 5(7), July 2004. 35

E. H. DURFEE, V. R. LESSER et D. D. CORKILL : Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, 1989. ISSN 1041-4347. 45

E. DURFEE et V. LESSER : Using Partial Global Plans to Coordinate Distributed Problem Solvers. *In Proc. of the International Joint Conference on Artificial Intelligence*, p. 875–883, August 1987. 46

E. DURFEE, V. LESSER et D. CORKILL : Coherent Cooperation Among Communicating Problem Solvers. *IEEE Transactions on Computers*, C36(11):1275–1291, November 1987. 45

E. DURFEE et V. LESSER : Negotiating Task Decomposition and Allocation Using Partial Global Planning. *Distributed Artificial Intelligence*, 2:229–244, 1989. 42

EAUCTIONHOUSE (WEB SITE) : http ://www.eauctionhouse.co.uk/. 54

EBAYS'S AUCTION (WEB SITE) : http ://pages.ebay.com. 54

EBXML : ebXML Technical Architecture Specification v1.0.4. Rap. tech., OASIS, 2001. http ://www.ebxml.org. 17

EBXML CPPA : ebXML Collaboration Protocol Profile and Agreement. Rap. tech., OASIS, 2002. 17

EDMUND H. DURFEE : *A Unified Approach to Dynamic Coordination : Planning actions and interactions in a distributed problem solving network*. Thèse de doctorat, Computer and Information Science, Univ. of Massachusetts., Sept. 1987. 46

T. ERL : *Service-Oriented Architecture (SOA) : Concepts, Technology, and Design*. Service-Oriented Computing Series, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2005. ISBN 0-13-185858-0. 140

P. FARATIN : *Automated Service Negotiation Between Autonomous Computational Agents*. Thèse de doctorat, Department of Electronic Engineering Queen Mary & WestField College, 2000. 55

P. FARATIN, C. SIERRA, N. JENNINGS, et P. BUCKLE : Designing Flexible Automated Negotiators : Concessions, Trade-Offs and Issue Changes. Rap. tech. RP-99-03, Institut d'Investigacio en Intel.ligencia Artificial Technical Report, 1999. 55

J. FERBER : *Les systèmes multi-agents : Vers une intelligence collective*. InterEditions, Paris., 1995. 50

C. FERNANDEZ, R. BEJAR, B. KRISHNAMACHARI et C. GOMES : Communication and computation in distributed csp algorithms. *In Proc. CP2002*, p. 664–679, Ithaca, NY, USA, July 2002. 47

R. B. FINDLER, M. LATENDRESSE et M. FELLEISEN : Object-oriented Programming Languages Need Well-founded Contracts. Rap. tech. TR 01-372, Rice University Computer Science, 2001. 14

T. FININ, J. WEBER, G. WIEDERHOLD, M. GENESERETH, D. MCKAY, R. FRITZSON, S. SHAPIRO, R. PELAVIN et J. MCGUIRE : Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993. 52

FIPA ORGANIZATION (WEB SITE) : http ://www.fipa.org/. 52

FIPA TC COMMUNICATION : FIPA ACL Message Structure Specification. Rap. tech., FIPA Organization, 2002a. 52

FIPA TC COMMUNICATION : FIPA Contract Net Interaction Protocol Specification. Rap. tech., FIPA Organization, 2002b. 54, 78

FISHMARKET (WEB SITE) : http ://www.iiia.csic.es/Projects/fishmarket/newindex.html. 54

S. FRANKLIN et A. GRAESSER : Is it an Agent, or Just a Program ? : A Taxonomy for Autonomous Agents. *In In Proc. of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages (ECAI)*, p. 21–35, London, UK, 1997. Springer-Verlag. ISBN 3-540-62507-0. 50

S. FRØLUND et J. KOISTINEN : QML : a language for quality of service specification. Rap. tech. HPL-98-10, Software Technology Laboratory, Hewlett-Packard, fév. 1998a. 30

S. FRØLUND et J. KOISTINEN : Quality of Service Aware Distributed Object Systems. Rap. tech. HPL-98-142, HP Software Technology Lab., 1998b. 30

S. FRØLUND et J. KOISTINEN : Quality-of-service specification in distributed object system. *Distributed System Engineering*, 5:179–202, 1998c. 30, 31

X. FU, W. SHI, A. AKKERMAN et V. KARAMCHETI : CANS : Composable, Adaptive Network Services Infrastructure. Rap. tech., New York, NY, USA, 2000. 36

D. GARLAN et B. SCHMERL : Model-based adaptation for self-healing systems. *In Proceedings of the first workshop on Self-healing systems (WOSS'2002)*, p. 27–32, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-609-9. 32

S. GÖBEL, C. POHL, R. AIGNER, M. POHLACK, S. RÖTTGER et S. ZSCHALER : The COMQUAD Component Container Architecture and Contract Negotiation. Rap. tech. TUD-FI04-04, Technische Universität Dresden, avr. 2004. 34

M. P. GEORGEFF : Communication and interaction in multi-agent planning. *Distributed Artificial Intelligence*, p. 200–204, 1988a. 46

M. P. GEORGEFF : A theory of action for multi-agent planning. *Distributed Artificial Intelligence*, p. 205–209, 1988b. 46

I. GEORGIADIS : *Self-Organising Distributed Component Software Architectures*. Thèse de doctorat, Imperial College, London, 2002. 32

H. GIESE : Contract-Based Component System Design. *In HICSS '00 : Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, p. 8051, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0493-0. 19

X. GU, D. WICHADAKUL et K. NAHRSTEDT : Visual qos programming environment for ubiquitous multimedia services. *In Proc. of IEEE International Conference on Multimedia and Expo 2001 (ICME2001)*, Tokyo, Japan, August 2001a. 30

X. GU, K. NAHRSTEDT, W. YUAN, D. WICHADAKUL et D. XU : An XML-based Quality of Service Enabling Language for the Web. Rap. tech., Champaign, IL, USA, 2001b. 30

R. GUTTMAN et P. MAES : Cooperative vs. competitive multi-agent negotiations in retail electronic commerce. *In CIA*, p. 135–147, 1998. 56

Y. HAMADI : *Traitement des problèmes de satisfaction de contraintes distribuées*. Thèse de doctorat, Université de Montpellier, July 1999. in French. 47

D. HAMLET, D. MASON et D. WOIT : Theory of Software Reliability Based on Components. *In ICSE'2001*. IEEE Computer, 2001. 97

J. HAN : *Temporal Logic Based Specifications of Component Interaction Protocols*, p. 43–52. Springer LNCS, 2000. 14

J. HAN : A Comprehensive Interface Definition Framework for Software Components. *In APSEC '98 : Proceedings of the Fifth Asia Pacific Software Engineering Conference*, p. 110, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-9183-2. 14, 19

G. T. HEINEMAN et W. T. COUNCILL, éds. *Component-based Software Engineering : Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0-201-70485-4. 1

R. HELM, I. HOLLAND et D. GANGOPADHYAY : Contracts : specifying behavioral compositions in object-oriented systems. *In OOPSLA/ECOOP '90 : Proceedings of the European conference on object-oriented programming on Object-oriented programming systems, languages, and applications*, p. 169–180, New York, NY, USA, 1990. ACM Press. ISBN 0-201-52430-X. 13

S. HISSAM, J. HUDAK, J. IVERS, M. KLEIN, M. LARSSON, G. MORENO, L. NORTHROP, D. PLAKOSH, J. STAFFORD, K. WALLNAU et W. WOOD : Predictable Assembly of Substation Automation Systems : An Experience Report. Rap. tech. CMU/SEI-2002-TR-031, CMU/SEI, 2002. 28

S. HISSAM, G. MORENO, J. STAFFORD et K. WALLNAU : Enabling Predictable Assembly. *Journal of Systems and Software*, 65(3), 2003. 97

Y. HOFFNER, S. FIELD, P. GREFEN et H. LUDWIG : Contract-driven creation and operation of virtual enterprises. *Comput. Networks*, 37(2):111–136, 2001. ISSN 1389-1286. 17

I. M. HOLLAND : Specifying Reusable Components Using Contracts. *In ECOOP '92 : Proceedings of the European Conference on Object-Oriented Programming*, p. 287–308, London, UK, 1992. Springer-Verlag. ISBN 3-540-55668-0. 13

I. HOLLAND : *The Design and Representation of Object-Oriented Components*. Thèse de doctorat, Northeastern University, Boston, 2003. 13

H. HÄRTIG, L. REUTHER, J. WOLTER, M. BORRISS et T. PAUL : Cooperating resource managers. *In Proceedings of Workshop on QoS Support for Real-Time Internet Applications*, Vancouver, Canada, June 1999. 34

M. HUHNS et M. SINGH, éds. *Readings in agents*. Morgan Kaufmann, San Francisco, 1998. 50

IEEE 1061 : IEEE 1061, Standard for a Software Quality Metrics Methodology. Rap. tech., The Institute of Electrical and Electronics Engineers, Inc., 1998. 23

ISO/IEC 9126 : ISO/IEC 9126, Software Product Evaluation - Quality Characteristics and Guidelines for their Use. Rap. tech., International Organization for Standarization. 23

JAMON PROJECT (WEB SITE), 2003 : http ://jamonapi.sourceforge.net/. 130

N. R. JENNINGS, S. PARSONS, C. SIERRA et P. FARATIN : Automated negotiation. *In Proc. of International Conference on the Practical Application of Intelligent Agents and Multi- Agent Systems (PAAM)*, p. 23–30, Manchester, UK, 2000. 42

N. JENNINGS, P. FARATIN, A. LOMUSCIO, S. PARSONS, C. SIERRA et M. WOOLDRIDGE : Automated negotiation : Prospects, methods and challenges. *Journal of Group Decision and Negotiation (GDN)*, 10(2):199 – 215, March 2001. 42

N. JENNINGS, P. FARATIN, T. NORMAN, P. O'BRIEN, M. E. WIEGAND, C. VOUDOURIS, J. L. ALTY, T. MIAH et E. H. MAMDANI : Adept : Managing business processes using intelligent agents. *In Proc. BCS Expert Systems 96 Conference (ISIP Track)*, p. 5–23, Cambridge, UK, 1996. 44

N. JENNINGS, M. WOOLDRIDGE et K. SYCARA : A roadmap of agent research and development. *International Journal of Autonomous Agents and Multi-Agent Systems*, 1:7– 38, 1998. 50

J.-M. JÉZÉQUEL et B. MEYER : Design by contract : The lessons of ariane. *Computer*, 30(1):129–130, 1997. ISSN 0018-9162. 14

J. JIN et K. NAHRSTEDT : Classification and Comparison of QoS Specification Languages for Distributed Multimedia Applications,. Rap. tech. UIUCDCS-R-2002-2302/UILU-ENG-2002-1745, Department of Computer Science, University of Illinois at Urbana-Champaign, 2002. 36

J. JIN et K. NAHRSTEDT : QoS Specification Languages for Distributed Multimedia Applications : A Survey and Taxonomy. *IEEE Multimedia Magazine*, 11(3):74–87, 2004. ISSN 1064-7570. 36

Y. JIN et J. HAN : Runtime Validation of Behavioural Contracts for Component Software. *In QSIC '05 : Proceedings of the Fifth International Conference on Quality Software*, p. 177–186, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2472-9. 20

A. KELLER, G. KAR, H. LUDWIG, A. DAN et J. HELLERSTEIN : Managing dynamic services : A contract based approach to a conceptual architecture. *In Proc. of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, April 15-19 2002. 17, 19

A. KELLER et H. LUDWIG : Defining and monitoring service level agreements for dynamic e-business. *In Proc. of the 16th USENIX Systems Administration Conference (LISA'02)*, Philadelphia, PA, November 3-8 2002. 19

A. KELLER et H. LUDWIG : The WSLA Framework : Specifying and Monitoring Service Level Agreements for Web Services. *J. Network and Systems Management*, 11(1):57–81, 2003. ISSN 1064-7570. 17

J. O. KEPHART : Research challenges of autonomic computing. *In ICSE '05 : Proceedings of the 27th international conference on Software engineering*, p. 15–22, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-963-2. 4

F. KON, M. ROMÁN, P. LIU, J. MAO, T. YAMANE, L. MAGALHÃES et R. CAMPBELL : Monitoring, security, and dynamic configuration with the dynamictao reflective orb. *In Proc. of IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, New York, April 3-7 2000. 2, 35

S. KRAUS : Automated Negotiation and Decision Making in Multiagent eEnvironments. *Multi-agents systems and applications*, p. 150–172, 2001. 18

Y. LABROU et T. FININ : A proposal for a new KQML specification. Rap. tech. TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250, Feb 1997. Also available online at http ://www.cs.umbc.edu/kqml/. 52

M. LARSSON : *Predicting Quality Attributes in Component-based Software Systems*. Thèse de doctorat, March 2004. 27

O. LAYAÏDA, S. B. ATALLAH et D. HAGIMONT : Reconfiguration-based QoS Management in Multimedia Streaming Applications. *In Proc. of the 30th IEEE/EUROMICRO Conference Track on "Multimedia & Telecommunications : Challenges in Distributed Multimedia Systems"*, Rennes, France, September 2004. 36

O. LAYAIDA et D. HAGIMONT : Designing self-adaptive multimedia applications through hierarchical reconfiguration. *In Proceedings of 5th IFIP WG 6.1 International Conference Distributed Applications and Interoperable Systems (DAIS'2005)*, vol. 3543 de *Lecture Notes in Computer Science*, p. 95–107. Springer, 2005. 36

M. LOHSE, M. REPPLINGER et P. SLUSALLEK : An Open Middleware Architecture for Network-Integrated Multimedia. *In IDMS/PROMS 2002 : Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems*, p. 327–338, London, UK, 2002. Springer-Verlag. ISBN 3-540-00169-7. 36

A. LOMUSCIO, M. WOOLDRIDGE et N. JENNINGS : A classification scheme for negotiation in electronic commerce. *In Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, p. 19–33, 2001. 57

O. LOQUES et A. SZTAJNBERG : Customizing Component-Based Architectures by Contract. *In Second International Working Conference on Component Deployment (CD 2004)*, vol. 3083 de *Lecture Notes in Computer Science*, p. 18–34, Edinburgh, UK, May 2004. Springer-Verlag. ISBN 3-540-22059-3. 33

S. LORCY, N. PLOUZEAU et J. JEZEQUEL : A Framework For Managing Quality of Service Contracts in Distributed Applications. *In Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*, p. 125–137. IEEE Computer Society, 1998. ISBN 0-8186-8482-8. 15

R. LUCE et H. RAIFFA : *Games and decisions*. John Wiley, New York, 1957. 47

H. LUDWIG, A. KELLER, A. DAN, K. KING et R. FRANCK : Web Service Level Agreement (WSLA) Language Specification, Version 1.0. Rap. tech., IBM Corporation, 2003. http ://www.research.ibm.com/wsla/WSLASpecV12003. 17

H. LUDWIG : A Conceptual Framework for Electronic Contract Automation. Rap. tech. RC 22608, IBM Research, 2002. 18

H. LUDWIG, H. GIMPEL, A. DAN et R. KEARNEY : Template-based automated service provisioning - supporting the agreement-driven service life-cycle. *In In Proceedings of ICSOC'2005 International Conference on Service-Oriented Computing*, vol. 3826 de *Lecture Notes in Computer Science*, p. 283–295. Springer, December 2005. ISBN 3-540-30817-2. 18

R. MALAN et D. BREDEMEYER : Defining Non-Functional Requirements. Rap. tech., White Paper, Bredemeyer Consulting, 2001. 26

T. MALONE et R. F. M. HOWARD : Enterprise : A market-like task scheduler for distributed computing environments. Rap. tech., Massachusetts Institute of Technology, Center for Information Systems Research Working, October 1983. 54

F. V. MARTIAL : Interactions among autonomous planning agents. *In* Y. DEMAZEAU et J.-P. MÜLLER, éds : *Decentralized A.I. : Proc. of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, p. 105–119. Elsevier Science, Amsterdam, 1990. 43

P. MATHIEU et M.-H. VERRONS : A generic model for Contract Negotiation. *In Proc. of the International Conference on Artificial Intelligence and the Simulation of Behaviour (AISB)*, London, UK, 3-5 April 2002. 58

P. MATHIEU et M.-H. VERRONS : Three Different Kinds of Negotiation Applications Achieved with GeNCA. *In Proc of International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA)*, 2004. ISBN 2-9599776-8-8. 59

D. MCILROY : *Mass-Produced of Software Components*. J.M. Buxton, P. Naur and B. Randell, editors, NATO Software Engineering Conference Concepts and Techniques, 1968. 1

N. MEDVIDOVIC et R. N. TAYLOR : A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions of Software Engineering*, 26(1):70–93, 2000. ISSN 0098-5589. 32

M. MERZ, F. GRIFFEL, H. WEINREICH et W. LAMERSDORF : Electronic contracting with COSMOS - How to establish, negotiate, and execute electronic contracts on the internet. *In Proc. of 2nd International Workshop on Enterprise Distributed Object Computing*, San Diego, November 1998a. 18

M. MERZ, F. GRIFFEL, M. T. TU, S. MULLER-WILKEN, H. WEINREICH, M. BOGER et W. LAMERSDORF : Supporting electronic commerce transactions with contracting services. *International Journal of Cooperative Information Systems*, 7(4):249–274, 1998b. 17

B. MEYER : Applying "Design by Contract". *IEEE Computer*, 25(10):40–51, oct. 1992a. 4, 13, 14, 20

B. MEYER : *Eiffel : the language*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992b. ISBN 0-13-247925-7. 14

S. MICHAEL et S. MARKUS : A Matchmaking Component for the Discovery of Agreement and Negotiation Spaces in Electronic Markets. *In Proc. Group Decision and Negotiation Conference*, p. 61–75, La Rochelle, France, 2001. 57

C. A. MINGINS et C. Y. CHAN : Building Trust in Third-Party Components Using Component Wrappers in the .NET Frameworks. *In CRPIT '02 : Proceedings of the Fortieth International Conference on Tools Pacific*, p. 153–157, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc. ISBN 0-909925-88-7. 20

MITRE CORPORATION (WEB SITE) : http ://www.mitre.org/. 23

K. MOAZAMI-GOUDARZI : *Consistency Preserving Dynamic Reconfiguration of Distributed Systems*. Thèse de doctorat, Imperial College, London, 1999. 32

D. G. A. MOBACH, B. J. OVEREINDER et F. M. T. BRAZIER : A resource negotiation infrastructure for self-managing applications. *In ICAC '05 : Proceedings of the Second International Conference on Automatic Computing*, p. 381–382, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7965-2276-9. 18

R. S. MOREIRA, G. S. BLAIR et E. CARRAPATOSO : A reflective component-based and architecture aware framework to manage architecture composition. *In Proceedings of the Third International Symposium on Distributed Objects and Applications (DOA'2001)*, p. 187, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1300-X. 32

H. J. MULLER : *Negotiation principles in Foundations of Distributed Artificial Intelligence G.M.P O'Hare, N.R Jennings*, chap. 7, p. 211–229. John Wiley and Sons, 1996. 51

M. MULUGETA et S. GÖBEL : Towards Distributed Contract Negotiation in Component-Based Systems. *In Proc. of Software Composition (SC)*, Edinburgh, Scotland, April 9 2005. 34

H. NWANA, D. NDUMU, L. LEE et J. COLLIS : Zeus : A tool-kit for building distributed mutli-agent systems. *Applied Artificial Intelligence Journal*, 13(1):129–186, 1999. 58

G. O'HARE et N. JENNINGS, éds. *Foundations of distributed artificial intelligence*. John Wiley & Sons, Inc., New York, NY, USA, 1996. ISBN 0-471-006750. 50

OMG : Object Constraint Language Specification. Rap. tech. version 1.1, ad/97-08-08, IBM `www.software.ibm.com/ad/ocl`, sept. 1997. 178

OPENFIRE SERVER (WEB SITE) : http ://www.jivesoftware.com/products/openfire/. 161

P. OREIZY, M. M. GORLICK, R. N. TAYLOR, D. HEIMBIGNER, G. JOHNSON, N. MEDVIDOVIC, A. QUILICI, D. S. ROSENBLUM et A. L. WOLF : An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14(3):54–62, 1999. ISSN 1541-1672. 3

M. J. OSBORNE et A. RUBINSTEIN : *A course in game theory*. MIT Press, Cambridge, MA, 1994. 47

M. OSBORNE : *An introduction to Game Theory*. Oxford University Press, New-York, 2004. 47

A. OZANNE : *(A paraître) Modèle général de contrat pour systèmes autonomes à base de composants logiciels*. Thèse de doctorat, Université Pierre et Marie Curie, Laboratoire d'informatique de Paris 6, 2007. 16

P. PAL, J. LOYALL, R. SCHANTZ, J. ZINKY, R. SHAPIRO et J. MEGQUIER : Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration. *In Proc. of ISORC 2000, 3rd IEEE International Symposium on Object-Oriented Real-time Distributed Computing*, Newport Beach, CA, March 15-17 2000a. 35

P. PAL, J. LOYALL, R. SCHANTZ, J. ZINKY, R. SHAPIRO et J. MEGQUIER : Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration. *In ISORC '00 : Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, p. 310, Washington, DC, USA, 2000b. IEEE Computer Society. ISBN 0-7695-0607-0. 30

M. PARASHAR et S. HARIRI : Autonomic computing : An overview. *In Unconventional Programming Paradigms*, p. 257–269. Springer LNCS, 2005. ISBN 978-3-540-27884-9. 4

D. PARNAS : On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, Décembre 1972. 1

W. PICARD : NeSSy : Enabling Mass E-Negotiations of Complex Contracts. *dexa*, 00:829, 2003. ISSN 1529-4188. 18

D. G. PRUITT : *Negotiation behavior*. Academic Press, New York, 1981. 5, 42

QUALITY OBJECTS (QUO) PROJECT (WEB SITE) : http ://quo.bbn.com/. 30, 35

A. RAUSCH : Software evolution in componentware using requirements/assurances contracts. *In ICSE '00 : Proceedings of the 22nd international conference on Software engineering*, p. 147–156, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-206-9. 15

REAL-TIME CORBA, 1997 : http ://www.cs.wustl.edu/ schmidt/corba-research-realtime.html. 35

B. REDMOND : *Supporting the Unanticipated Dynamic Adaptation of Object-Oriented Software*. Thèse de doctorat, Trinity College, Dublin, 2003. 33

R. REUSSNER : Enhanced component interfaces to support dynamic adaption and extension. *In HICSS '01 : Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 9*, p. 9043, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0981-9. 15, 19

R. REUSSNER : Automatic component protocol adaptation with the CoConut/J tool suite. *Future Generation Computer Systems, Tools for Program Development and Analysis*, 19:627–639, 2003. 15

R. H. REUSSNER, H. W. SCHMIDT et I. H. POERNOMO : Reliability Prediction for Component-Based Software Architectures. *Journal of Systems and Software*, 66(3):241–252, 2003. ISSN 0164-1212. 140

J. RODRÍGUEZ-AGUILAR, F. MARTÍN, P. NORIEGA, P. GARCIA et C. SIERRA : Towards a test-bed for trading agents in electronic auction markets. *Artificial Intelligence Communications*, 11:5–19, 1998. 57

J. ROSENSCHEIN et G. ZLOTKIN : *Rules of Encounter : Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994. 47, 55

S. RÖTTGER et S. ZSCHALER : CQML+ : Enhancements to CQML. *Proc. 1st Intl. Workshop on Quality of Service in Component-Based Software Engineering (QoSCBSE'2003), Cépaduès- Éditions*, p. 43–56, June 2003. Toulouse, France. 32

T. SANDHOLM et V. LESSER : Issues in automated negotiation and electronic commerce : Extending the contract net framework. *In* V. LESSER, éd. : *Proc. of the International Conference on Multi-Agent Systems (ICMAS'95)*, p. 328–335, San Francisco, CA, USA, 1995. The MIT Press : Cambridge, MA, USA. 54

T. W. SANDHOLM : An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. *In Proceedings of the 12th International Workshop on Distributed Artificial Intelligence*, p. 295–308, Hidden Valley, Pennsylvania, 1993. 54

M. SAYAL, A. SAHAI, V. MACHIRAJU et F. CASATI : Semantic analysis of e-business operations. *Journal of Network and Systems Management*, 11(1):13–37, 2003. ISSN 1064-7570. 19

D. SCHMIDT : *Design Patterns in Communication Software*, chap. 18, Applying a Pattern Language to Develop Extensible ORB Middleware, p. 393–418. SIGS reference library series 19. Cambridge University Press, Laura Rising édn, 2001. 35

D. C. SCHMIDT et F. KUHNS : An overview of the Real-time CORBA specification. *IEEE Computer special issue on Object-Oriented Real-time Distributed Computing*, June 2000. 35

P. SHARMA, J. LOYALL, G. HEINEMAN, R. SCHANTZ, R. SHAPIRO et G. DUZAN : Component-based dynamic qos adaptations in distributed real-time and embedded systems. *In Proc. of International Symposium on Distributed Objects and Applications (DOA'04)*, Agia Napa, Cyprus, October 25-29 2004. 35

C. SIERRA, P. FARATIN et N. JENNINGS : A Service-Oriented Negotiation Model Between Autonomous Agents. *In* M. BOMAN et W. V. de VELDE, éds : *Proc. of European Workshop on Modeling Autonomous Agents in Multi-Agent World*, num. 1237 de Lecture Notes in Artificial Intelligence, p. 17–35. Springer-Verlag, 1997. 55

J. SKENE, D. D. LAMANNA et W. EMMERICH : Precise service level agreements. *In ICSE '04 : Proceedings of the 26th International Conference on Software Engineering*, p. 179–188, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2163-0. 18

R. G. SMITH : The Contract Net Protocol : High Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 12(C-29):1104–1113, 1980. 42, 45, 53

R. G. SMITH et R. DAVIS : *Frameworks for cooperation in distributed problem solving*, p. 61–70. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1980. ISBN 0-934613-63-X. 42, 54

N. L. SOMMER : *Contractualisation des ressources pour les composants logiciels : une approche réflexive*. Thèse de doctorat, Université de Bretagne Sud, Vannes France, Décembre 2003. 33

J. STAFFORD et J. MCGREGOR : Issues in Predicting the Reliability of Components. *In ICSE'2002 CBSE Workshop*, 2002. 97

M. STRÖBEL : Design of roles and protocols for electronic negotiations. *Electronic Commerce Research, Special Issue on Market Design*, 1(3):335–353, 2001. 57

C. SZYPERSKI : *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. ISBN 0201745720. 1

UNIFRAME PROJECT (WEB SITE) : http ://www.cs.iupui.edu/uniFrame/. 24

M.-H. VERRONS : *GeNCA : un modèle général de négociation de contrats entre agents*. Thèse de doctorat, Université des Sciences et Technologies de Lille, 2 novembre 2004. 58

D. G. WADDINGTON et G. COULSON : A distributed multimedia component architecture. *In EDOC '97 : Proceedings of the 1st International Conference on Enterprise Distributed Object Computing*, p. 334, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8031-8. 36

G. WAGNER : Artificial agents and logic programming. *In Panel statement at the ICLP'97 post-conference workshop Logic Programming and Multiagent Systems*, p. 22, 1997. 53

K. WALLNAU, J. STAFFORD, S. HISSAM et M. KLEIN : On the Relationship of Software Architecture to Software Component Technology. *In ECOOP'2001 WCOP Workshop*, 2001. 97

S. WEERAWARANA, F. LEYMANN, F. CURBERA, T. STOREY et D. F. FERGUSON : *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging*. Pearson Education, 2005. ISBN 0131488740. 140

S. R. WHITE, J. E. HANSON, I. WHALLEY, D. M. CHESS et J. O. KEPHART : An Architectural Approach to Autonomic Computing. *Proceedings of International Conference on Autonomic Computing (ICAC'2004)*, 00:2–9, 2004. 32

S. WOLLKIND, J. VALASEK et T. IOERGER : Automated conflict resolution for air traffic management using cooperative multiagent negotiation. *In AIAA Guidance, Navigation, and Control Conference*, Providence, Rhode Island, August 16-19 2004. 44

P. WURMAN, M. WELLMAN et W. WALSH : The Michigan Internet AuctionBot : A Configurable Auction Server for Human and Software Agents. *In* K. P. SYCARA et M. WOOLDRIDGE, éds : *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, p. 301–308, New York, 9–13, 1998. ACM Press. ISBN 0-89791-983-1. 54, 57

S. ZSCHALER et M. MEYERHÖFER : Explicit modelling of QoS-Dependencies. *Proc. 1st Intl. Workshop on Quality of Service in Component-Based Software Engineering (QoSCBSE'2003), Cépaduès-Éditions*, p. 57–66, June 2003. Toulouse, France. 32

A

Cas d'étude et illustrations

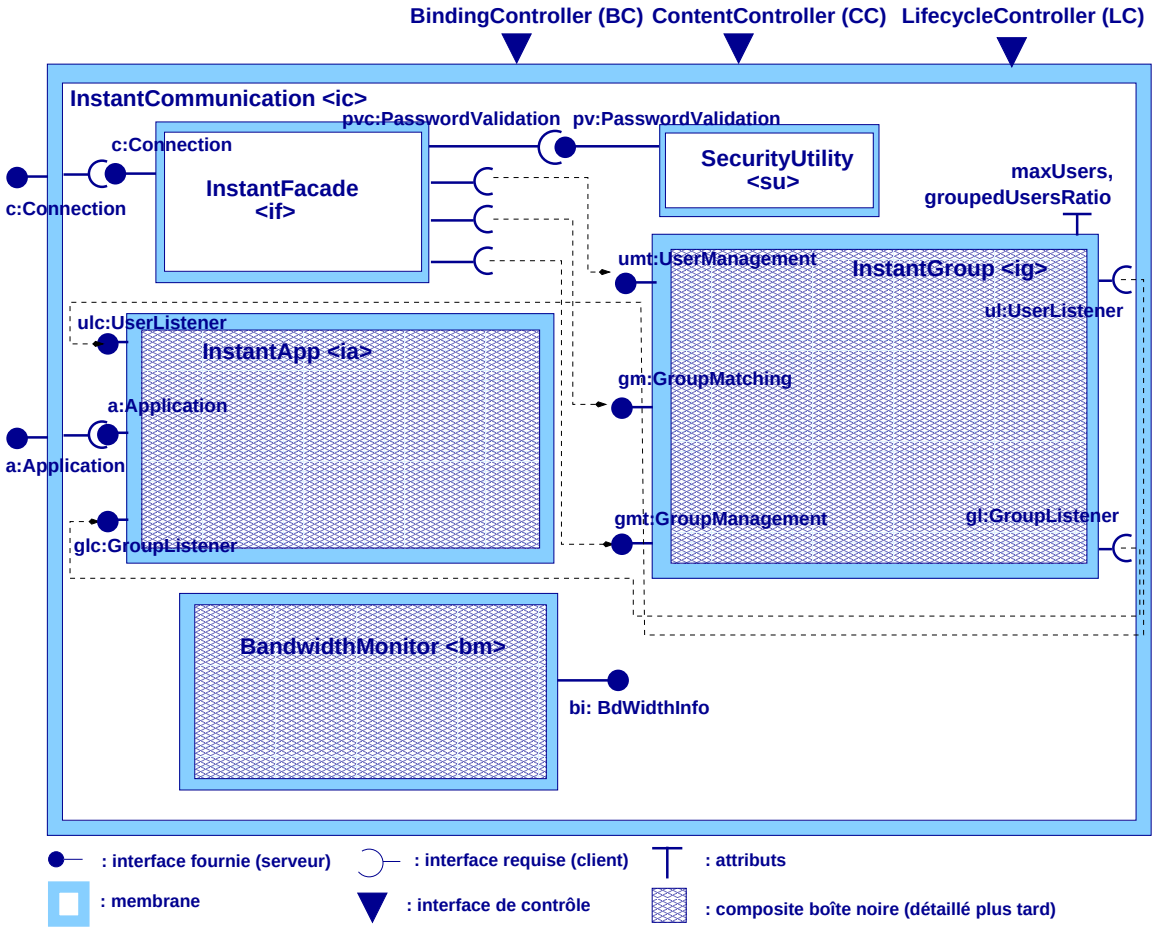
Dans cette annexe, nous présentons les illustrations complémentaires sur le modèle de négociation. L'architecture de l'application considérée est tout d'abord très brièvement décrite, puis les illustrations complémentaires des contrats et de scénarii de négociation sont donnés.

Le cas d'étude considéré consiste en une application distribuée de communication instantanée. La fonctionnalité principale de cette application consiste à former automatiquement des groupes d'utilisateurs et de les faire partager automatiquement diverses applications en fonction de leurs centres d'intérêts. Ainsi, un utilisateur donné se connecte au système en fournissant des informations le concernant (login, mot de passe) ainsi qu'une liste de mot-clés décrivant ses centres d'intérêts. Le système trouve alors automatiquement les groupes dont les thèmes correspondent aux centres d'intérêts de l'utilisateur, et y ajoute ce dernier. Ainsi, à l'intérieur d'un même groupe, les utilisateurs peuvent alors partager diverses applications qui correspondent aux thématiques du groupe et qui sont adaptées à leurs centres d'intérêts (messagerie instantanée, service vidéo...). Cette application distribuée est intégrée au serveur de messagerie instantané *Openfire* [Openfire Server (Web Site)], basé sur le protocole *XMPP*, et dont l'architecture ouverte permet l'extension à de nouvelles fonctionnalités. L'application est hébergée dans un serveur d'application web *Apache Tomcat*.

A.1 Architecture

L'architecture générale de l'application est décrite à la figure A.1. Le composant principal `Instant-Communication` représente l'application. D'un point de vue externe, il offre l'interface *Fractal* `c :Connection` de nom `c` et de signature `Connection`. La méthode `connectUser()` de cette interface représente le point d'entrée de l'application en permettant de connecter un utilisateur au système en lui fournissant certaines informations (login, mot de passe, centres d'intérêts). L'architecture interne de l'application (cf. Fig. A.1) est composée des cinq sous-composants suivants :

– le composant `InstantFacade` a en charge le pilotage de l'ensemble des fonctionnalités de gestion des utilisateurs et des groupes. Il offre le service de connexion au système au travers de la méthode `connectUser()` de l'interface `c :Connection`, et requiert les interfaces de `InstantGroup` de la façon suivante : *(i)* `umt :UserManagement` pour créer et enregistrer



Signatures d'interface

```
interface Connection{

 int connectUser(String username, String password,
                 Collection<String> keywords);
...
}
```

```
interface PasswordValidation{

 boolean isPasswordValid(String password);
...
}
```

```
interface GroupManagement{

 Collection<String> getGroupsId();
 Collection<String> getGroupTopics(String groupid);
 int addUserToGroups(User u, Collection<String> groupsid);
...
}
```

```
interface GroupMatching{

 Collection<String> searchGroupForKeywords(Collection<String>
                                           keywords);
...
}
```

```
interface UserManagement{

 User createUser(String username,
                 String password);
 int registerUser(User u);
 User getUser(String username);
...
}
```

```
interface Application{

 int start(String id);
 int sendMessage(String id,
                 String msg);
 String getNextMessage(String id);
...
}
```

```
interface BdWidthInfo{

 int getBdWLevel();
...
}
```

FIGURE *A.1 – Architecture générale de l'application*

les utilisateurs, *(ii)* `gm :GroupMatching` pour obtenir la liste des identifiants des groupes dans lesquels ajouter les utilisateurs, suite à l'appariemment entre les centres d'intérêts des utilisateurs et les thèmes des groupes, et *(ii)* `gmt :GroupManagement` pour demander l'ajout des utilisateurs dans des groupes donnés.
- le composant `SecurityUtility` offre des fonctionnalités utilitaires relatives à la sécurité comme pour, par exemple, valider la robustesse d'un mot de passe (méthode `isPasswordValid()` de l'interface `pv :PasswordValidation`).
- le composite `InstantGroup` permet de gérer tout ce qui concerne les utilisateurs et les groupes. D'un point de vue externe, il offre trois interfaces `umt :UserManagement`, `gm :GroupMatching` et `gmt :GroupManagement` pour, respectivement, gérer les utilisateurs, effectuer l'appariemment entre les centres d'intérêts des utilisateurs et les thèmes des groupes, et gérer les groupes. Les interfaces requises `un :UserListener` et `gn :GroupListener` servent à notifier le composant `InstantApp` des événements pertinents sur les utilisateurs et les groupes, selon un patron de conception *Observer* [15]. L'architecture interne de ce composant est décrite ci-après.
- le composite `InstantApp` permet de gérer tout ce qui concerne les applications partagées par les utilisateurs à l'intérieur de leurs groupes. L'architecture interne de ce composant est décrite dans un paragraphe qui suit.
- le composant `BandwidthMonitor` représente un composant de sonde de ressources qui permet d'observer certains paramètres liés à la bande passante. Il fournit par la méthode `getBdWLevel()` de l'interface `bi :BdWidthInfo` la consommation de bande passante globale de l'application. L'architecture interne de ce composant est décrite dans un paragraphe qui suit.

A.2 Contrats et négociations

Dans cette section, nous présentons les exemples de clauses sur trois contrats de composition externe, un contrat de composition interne et un contrat d'interface. Pour chaque clause de contrats, les éléments du modèle de négociation sont brièvement rappelés, et des scénarii de négociation qui exploitent les deux politiques, par concession et par effort, sont décrits. Pour être plus concis, les contrats et les composants concernés sont directement présentés pour chaque contrat. Les architectures détaillées des composants sont regroupées dans la section A.3 de cette annexe, page 174.

Le bilan des résultats en sortie de négociation est le suivant :
- le premier contrat de composition externe porte notamment sur des propriétés de configuration. Sa négociation aboutit, par la politique par concession, au maintien de la clause et, par la politique par effort, à modifier un attribut du composant garant (cf. page 164) ;
- le deuxième contrat de composition externe aboutit, par la politique par concession, à maintenir ou relâcher la clause, et conduit, par la politique par effort, à un échec (cf. page 166) ;
- le troisième contrat de composition externe spécifie diverses propriétés fonctionnelles d'exécution, pour lesquelles il n'y a que très peu de possibilité de négociation (cf. page 167) ;
- le contrat interne contient deux clauses qui portent sur des propriétés de fonctionnement et de ressources. Leurs négociations par la politique par concession aboutit à remplacer ou retirer entièrement la clause, et par la politique par effort, à propager la négociation vers les composants qui implémentent les propriétés négociées (cf. page 169) ;
- le contrat d'interface contient une clause spécifiant des paramètres d'appels de méthodes et pour laquelle les négociations sont limitées (cf. page 172) .

15. L'inscription des écouteurs se fait lorsque les composants se connectent à traver le *BindingController*.

A.2.1 Contrat externe sur le composant <umgr>

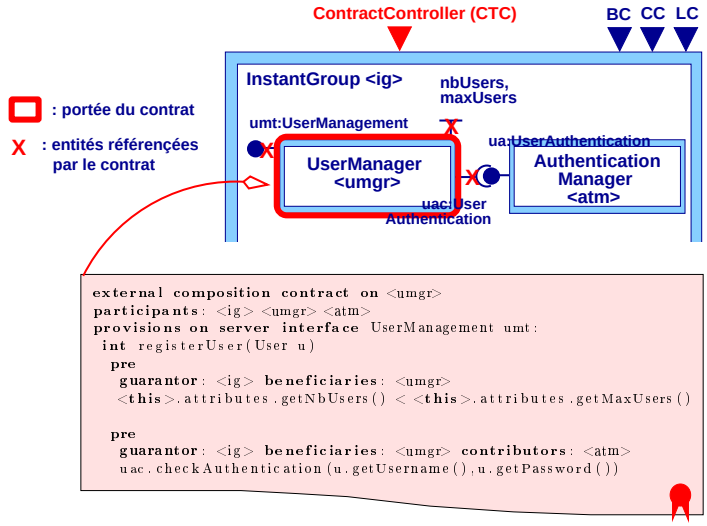


FIGURE A.2 – *Contrat de composition externe sur <umgr>*

L'architecture complète du composant `InstantGroup` est décrite à la page 174.

Clause sur le nombre maximal d'utilisateurs Cette première clause vérifie, à l'entrée de la méthode d'enregistrement `registerUser`, que le nombre d'utilisateurs présents dans le système est en deçà du seuil maximum. Elle exprime donc un seuil de charge, en terme de nombre d'utilisateurs acceptables dans le système, et doit être vérifiée à l'exécution du système. Cette précondition est violée lorsqu'à l'enregistrement d'un nouvel utilisateur, le nombre d'utilisateurs maximal est déjà atteint (le système tente l'enregistrement du 101^{eme} utilisateur alors que `maxUsers`$= 100$). Dans ce cas, une négociation atomique démarre, et fait intervenir comme parties le gestionnaire de contrats de `<ig>` comme initiateur, le composant bénéficiaire `<umgr>` et le composant garant `<ig>`.

Politique par relâchement. En négociant selon la politique par relâchement, les parties négociantes sont le gestionnaire de contrats de `<ig>` et `<umgr>`. Nous supposons que le gestionnaire de contrats de `<ig>` demande alors la négociabilité à `<umgr>` lequel accepte car étant équipé de sa liste d'alternatives suivante : $\mathcal{A}_{pre,<umgr>} := \{\text{STOP}\}$. De cette manière, à la première demande de relâchement, `<umgr>` répond par STOP, d'après sa liste d'alternatives, pour refuser le retrait de la clause, puisqu'elle représente une contrainte forte de tolérance du système. Comme `<umgr>` est l'unique bénéficiaire, la clause est conservée et la négociation s'achève en échec pour cette politique.

Politique par effort. Pour la politique par effort, les parties négociantes sont le gestionnaire de contrats de `<ig>` et `<ig>` lui-même. Pour démarrer la négociation, le gestionnaire de contrats de `<ig>` demande alors la négociabilité à `<ig>` (voir Fig. A.3). Nous partons ici du principe que, comme `<ig>` a la possibilité de propager la négociation de cette clause il accepte alors la négociation par effort. En effet, le garant `<ig>` est l'englobant qui est responsable de l'usage de son sous-composant `<umgr>` sur lequel la propriété `maxUsers` est réalisée, il a ainsi la possibilité de propager la négociation pour consulter `<umgr>`. Ainsi, à la demande d'effort formulée par l'initiateur, `<ig>` propose de propager la négociation (*étape 1*). Son gestionnaire de contrats

prend donc en charge la négociation et consulte le composant <umgr> (*étape 2*). Le gestionnaire de contrats de <ig> s'adresse à <umgr> pour obtenir des propositions d'efforts visant à revalider la clause négociée. Par la suite, en attribuant des capacités de négociation comme $\mathcal{A}_{pre,<umgr>} := \{\text{maxUsers} \leftarrow 250, \text{STOP}\}$, <umgr> peut proposer des modifications sur le terme maxUsers de la clause. Si la clause est satisfaite, la négociation atomique s'achève par un succès. Sinon, le gestionnaire de contrats annule la modification faite et redemande un effort auquel <umgr> répond par STOP pour signifier la fin de ses efforts. La propagation s'arrête alors, et remonte au niveau de l'initiateur initial de la négociation (ici le gestionnaire de contrats de <ig>), soit pour qu'il propose un code d'adaptation portant sur son contenu, soit pour qu'il signifie l'arrêt avec échec de la négociation par effort.

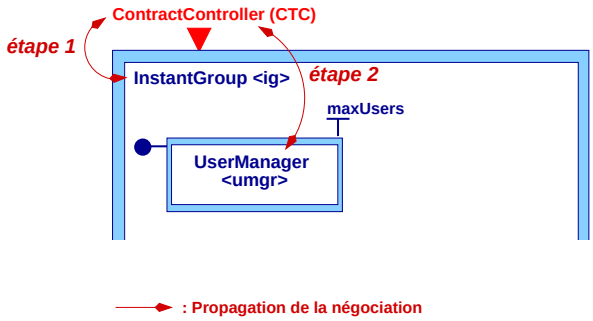


FIGURE *A.3 – Schéma de propagation pour la propriété MaxUsers*

Pour remarque, la négociation atomique, dans cet exemple, conduit, dans les deux politiques, à consulter le composant <umgr>. Toutefois les processus de négociation sont fondamentalement différents. Dans la politique par relâchement, <umgr> est directement consulté dans son rôle de bénéficiaire et indique l'arrêt de la négociation tout en conservant la clause. Dans la politique par effort, <umgr> est plus indirectement consulté. Il intervient après propagation proposée par le garant <ig>, car il réalise la propriété maxUsers spécifiée dans la clause. <umgr> aurait aussi très bien pu ne pas être consulté, par exemple si <ig> avait décidé lui-même, sans propagation, de re-satisfaire la propriété par des actions de négociation dans sa portée (reconfiguration du composant <umgr>, modification directe d'attribut de <umgr>, etc.).

Clause sur l'authentification des utilisateurs Cette précondition du contrat vérifie à l'entrée de la méthode d'enregistrement que l'utilisateur correspondant au couple (username,password) est bien authentifié. Elle est vérifiée à l'exécution car elle dépend des valeurs effectives des paramètres username et password pour un utilisateur donné, et est violée dès qu'un utilisateur tente de s'enregistrer dans le système sans qu'il ait été préalablement crée. Dans ce cas, une négociation atomique démarre avec les mêmes parties et les mêmes responsabilités, que pour la clause précédente. Le gestionnaire de contrats de <ig> dans le rôle de l'initiateur consulte le composant bénéficiaire <umgr> dans la politique par relâchement, et le composant garant <ig> dans la politique par effort. En revanche, bien que la clause concerne un service technique, comme celui-ci est particulièrement sensible et important, nous considérons que la négociation de la clause va être limitée (ou du moins la négocier, ne peut se résumer qu'à un retrait de celle-ci). Dans la politique par relâchement, le bénéficiaire principal accepte la négociation et propose l'alternative STOP pour indiquer le refus de relâchement avec maintien de la clause. Dans la politique par effort, le garant refuse directement la négociation car il est dans l'impossibilité de proposer des efforts pour revalider la clause (erreur de fonctionnement du composant <umgr>, voire <atm>,

ou du niveau de l'utilisateur.). La négociation se termine en échec pour les deux politiques. De plus, comme cette violation peut remettre en cause l'intégrité du système sans modifications fonctionnelles apparentes *a priori*, elle est d'importance au moins égale aux erreurs fonctionnelles ; la signalisation d'une telle violation de contrat doit alors être faite au plus haut-niveau.

A.2.2 Contrat externe sur le composant <if>

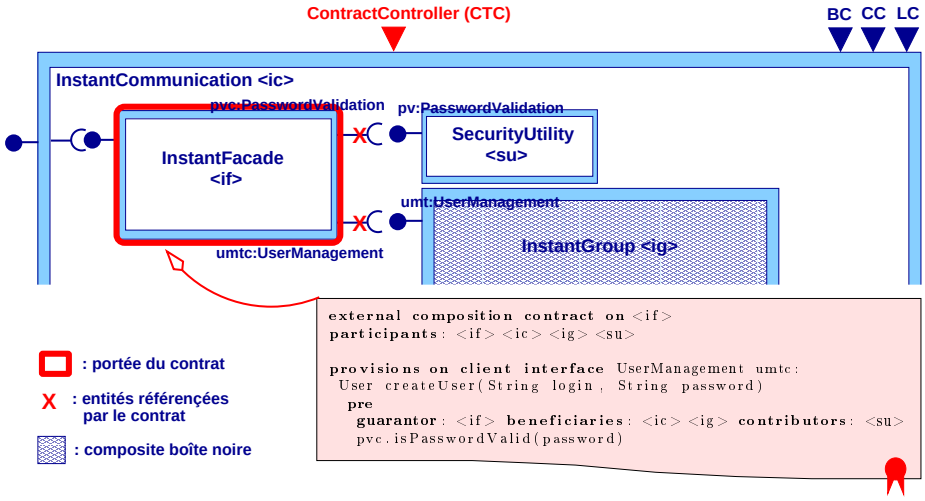


FIGURE A.4 – *Contrat de composition externe sur <if>*

Clause sur la robustesse du mot de passe La clause du contrat de composition externe (cf. Fig. A.4) exprime une précondition sur la méthode `createUser()` de l'interface requise `umtc :UserManagement` de sorte qu'avant l'appel de la méthode `createUser`, le composant `<if>` utilise le service de l'interface fournie `pv :PasswordValidation`, pour vérifier la robustesse du mot de passe. Cette clause est vérifiée à l'exécution du système puisqu'elle concerne la valeur effective du paramètre `password` à l'appel de la méthode `isPasswordValid`. Si le mot de passe n'est pas suffisamment robuste, la clause est violée et une négociation atomique démarre et les composants impliqués sont le gestionnaire de contrats de `<ic>` dans le rôle de l'initiateur, les composants bénéficiaires `<ic>` et `<ig>`, et le composant garant `<if>`.

Politique par relâchement. Dans la politique par relâchement, les parties négociantes sont le gestionnaire de contrats de `<ic>` dans le rôle de l'initiateur et, les composants `<ic>` et `<ig>` en tant que bénéficiaire principal et secondaire. Le gestionnaire de contrats demande tout d'abord la négociabilité de la clause aux deux bénéficiaires, et en fonction des réponses de `<ic>` et `<ig>` et des poids d'importance qui leur sont accordés, il évalue la négociabilité effective de la clause. Si la clause n'est pas négociable alors la négociation s'arrête pour cette politique. Sinon, le gestionnaire de contrats consulte alors le bénéficiaire principal `<ic>` qui négocie seul au nom des deux bénéficiaires. Comme il s'agit de l'englobant et qu'il est bénéficiaire de la correcte utilisation du composant `<if>`, nous supposons qu'il peut alors proposer le retrait complet de la clause, avec l'alternative RELEASE, pour exprimer son consentement à se passer de la vérification du mot de passe dans l'utilisation du sous-composant `<if>`. Ainsi, comme le seul relâchement proposé ici consiste en une demande de retrait, l'initiateur consulte alors l'ensemble des bénéficiaires `<ic>`

et <ig>, pour leur demander le retrait définitif. <ig> est maintenant consulté pour le retrait car, comme il fournit le service connecté à l'interface umtc, il est bénéficiaire de l'invocation de la méthode createUser. Autant il n'a pas la visibilité pour proposer des modifications au cours de la négociation, autant son rôle de fournisseur d'un service requis par <if> lui donne le droit d'être consulté au moment du retrait. À partir des réponses et des poids d'importance de <ic> et <ig>, le gestionnaire de contrats évalue alors le retrait définitif de la clause et la négociation s'achève avec succès si le retrait est autorisé et à bien lieu, ou échec sinon.

Pour remarque, de la même façon que la clause du contrat d'interface qui vérifie le nom d'utilisateur, le bénéficiaire principal <ic> pourrait très bien proposer, dans des contextes particuliers, l'arrêt de la négociation avec maintien de la clause, s'il requiert obligatoirement un certain niveau de robustesse des mots de passes. Dans ce cas, <ig> proposerait l'alternative STOP, et la négociation s'achèverait en échec.

Politique par effort. Dans la politique par effort, le gestionnaire de contrats de <ic> consulte le composant garant <if> qui, en tant que responsable de l'invocation de la méthode createUser, doit satisfaire la contrainte sur le mot de passe. Ainsi, le gestionnaire de contrats demande au garant <if>, la négociabilité de cette clause. De même que pour la clause du contrat d'interface, comme cette clause porte précisément sur l'invocation d'une méthode par le composant garant, nous supposons que le composant garant ne peut que proposer le refus de la négociation. <if> ne satisfait pas la clause en l'état ; il ne peut ni proposer des actions de négociation, ni propager vers de nouveaux composants. La négociation par effort s'achève alors par un échec.

A.2.3 Contrat externe sur le composant <gmtr>

L'architecture complète du composant InstantGroup est décrite à la page 174.

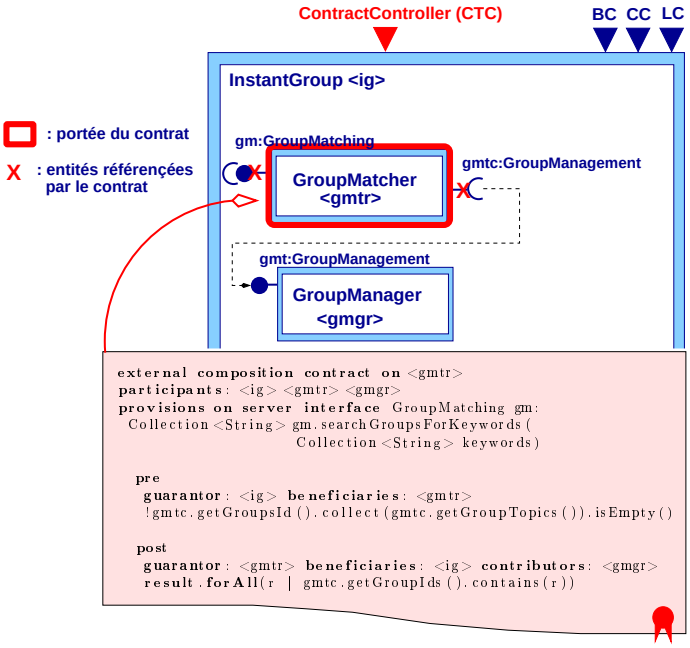


FIGURE A.5 – *Contrat de composition externe sur <gmtr>*

Clause sur l'existence d'une liste de thèmes non vide La première clause du contrat de composition externe sur le composant `<gmtr>` (précondition dans la Fig. A.5) établit une contrainte à l'entrée de la méthode `searchGroupsForKeywords`. Elle exprime une condition nécessaire à l'appariemment qui consiste à vérifier à l'entrée de la méthode qu'il existe au moins un groupe avec sa liste de thèmes non vide, de sorte que les centres d'intérêts (keywords) puissent être comparés à un ensemble non vide de thèmes (topics). Cette clause est vérifiée à l'exécution du système car elle concerne des éléments d'exécution des groupes (ensemble des groupes existants, thèmes associés, etc.). Cette clause est violée si il n'existe pas au moins un groupe dont la liste des thèmes est non vide. Dans ce cas, une négociation atomique démarre avec comme parties négociantes : le gestionnaire de contrats de `<ig>` dans le rôle de l'initiateur, le composant `<gmtr>` en tant que bénéficiaire et le composant `<ig>` en tant que garant.

Politique par relâchement. Dans la politique par relâchement, les parties négociantes sont le gestionnaire de contrats de `<ig>` et `<gmtr>`. Nous partons du principe que, suite à la demande de négociabilité de l'initiateur, `<gmtr>` accepte la négociation de cette clause, et propose le maintien de la clause avec l'alternative STOP car cette précondition est nécessaire à son usage afin qu'il puisse effectuer des appariements. Comme `<ig>` est le seul bénéficiaire, la clause est maintenue telle quelle, et la négociation par relâchement s'arrête en échec.

Politique par effort. Avec la politique par effort, les parties négociantes sont maintenant le gestionnaire de contrats de `<ig>` et `<ig>` lui-même. Du point de vue de `<ig>`, composant englobant, comme la violation consiste en une erreur liée à des éléments d'exécution (ici l'état des groupes), `<ig>` ne peut pas négocier par effort sans autre moyen de raisonnement. La négociation s'achève alors par un échec.

Clause sur la validité des groupes trouvés La seconde clause du contrat établit une contrainte à la sortie de la méthode `searchGroupsForKeywords` (cf. postcondition sur la Fig. A.5). Elle vérifie à la sortie de la méthode, que chaque groupe rendu fait bien partie de l'ensemble des groupes existants, et est vérifiée à l'exécution du système car elle concerne des éléments d'exécution (ensemble des groupes existants, thèmes associés, etc.). Cette clause est violée si les groupes rendus ne font pas partie de l'ensemble des groupes existants. Dans ce cas, une négociation atomique démarre avec comme parties négociantes : le gestionnaire de contrats de `<ig>` dans le rôle de l'initiateur, avec cette fois-ci, le composant bénéficiaire `<ig>` et le composant garant `<gmtr>`.

Bien que les parties négociantes soient différentes, les propriétés spécifiées concernent aussi des éléments d'exécution (cohérence entre les groupes trouvés suite à l'appariement et ceux existants), et les possibilités de négociation sont donc aussi limitées que pour la clause précédente.

Politique par relâchement. Dans la politique par relâchement, le composant `<ig>` est consulté. Nous partons du principe que ce bénéficiaire ne peut pas proposer de relâchements, car la clause entière lui est donc nécessaire, et lui permet bénéficier de la correcte utilisation du composant `<gmtr>` vis-à-vis de `<gmgr>`. `<ig>` accepte donc la négociation et propose le maintien de la clause avec l'alternative STOP. La négociation par relâchement s'achève par un échec.

Politique par effort. Pour la politique par effort, c'est le composant `<gmtr>` qui est consulté. Comme cette clause vérifie le bon fonctionnement de `<gmtr>`, celui-ci n'est vraisemblablement pas prévu pour négocier de telles violations. Cette violation relève davantage d'une erreur de fonctionnement de `<gmtr>`, et ce dernier ne peut alors pas proposer d'effort qui contribuerait à revalider la clause. La clause n'est pas négociable par une politique par effort, et la négociation se termine par un échec.

Comme ce contrat de composition externe spécifie l'usage fonctionnel du composant `<gmtr>`, des contraintes sont établies sur des éléments qu'il requiert à l'entrée de sa méthode (l'existence de thèmes

non vides), ainsi que sur les éléments rendus (les groupes trouvés font partie de ceux existants). Ainsi, comme ces éléments concernent le fonctionnement des composants, il n'y a que très peu de possibilités d'agir lorsque des erreurs portent sur de tels éléments, et ce indépendamment des mécanismes envisagés. Ces violations qui traduisent des erreurs de fonctionnement ne sont pas tant vouées à être rattrapées et gérées, qu'à être détectées et signalées au plus haut-niveau.

A.2.4 Contrat interne sur le composant <ic>

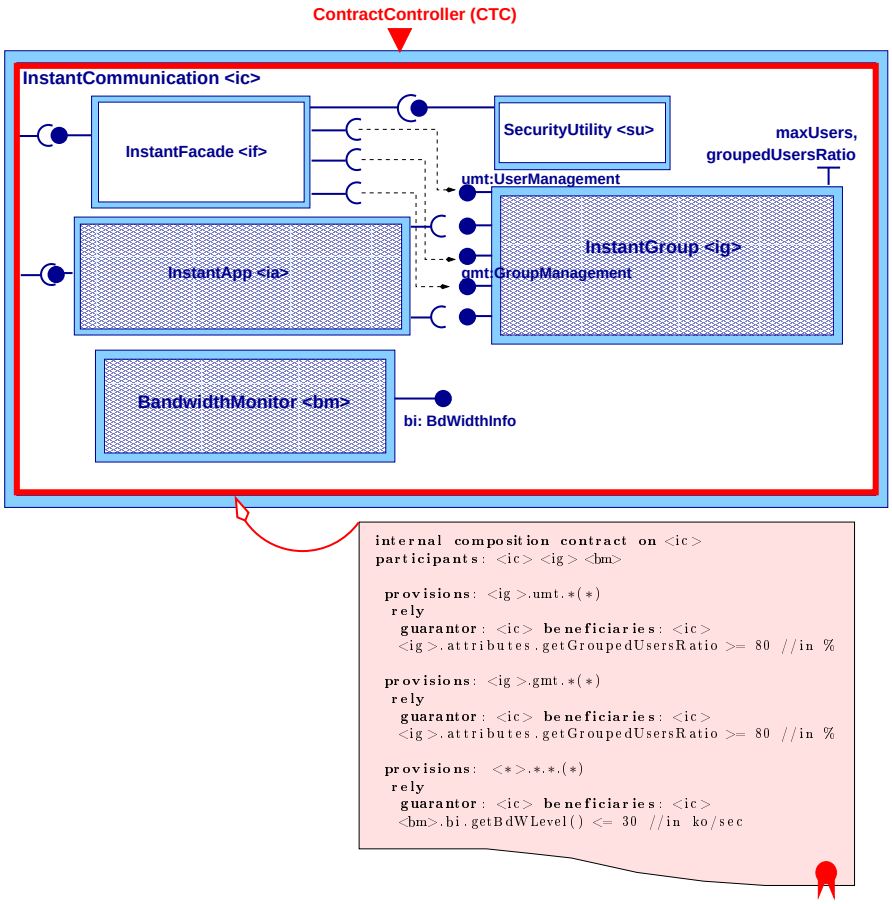


FIGURE A.6 – *Contrat de composition interne sur <ic>*

Les trois clauses du contrat de composition interne de `<ic>` (cf. Fig. A.6) décrivent les aspects du fonctionnement du composant `<ic>` (taux d'utilisateurs affectés dans des groupes, consommation de bande-passante). Les architectures complètes des composants `InstantGroup`, `InstantApp` et `BandwidthMonitor` sont décrites aux pages 174, 175 et 175.

Clause sur le taux d'affectation des utilisateurs Les deux premières clauses doivent être vérifiées des exécutions de toutes les méthodes des interfaces `<umt>` et `<gmt>` du composant `<ig>` (la construction *rely* spécifie la portée temporelle et le motif $\star$ indique une portée spatiale arbitraire). Elles expriment le fait que le ratio entre le nombre d'utilisateurs enregistrés dans le système et le nb d'utilisateurs répartis dans les groupes doit être supérieur à 80%. Elles sont violées si, par exemple, le taux d'utilisateurs

affectés (`groupedUsersRatio`) est inférieur à $80\,\%$ (ie. au moins 2 utilisateurs sur 10 sont connectés au système sans être affectés dans des groupes). Les parties impliquées dans la négociation atomique sont les mêmes que dans la clause précédente : le gestionnaire de contrats de <ic> dans le rôle de l'initiateur, et <ic> lui-même dans le rôle de composant à la fois garant et bénéficiaire.

Politique par relâchement. Dans la politique par relâchement, le gestionnaire de contrats de <ic> demande tout d'abord à ce dernier la négociabilité de la clause. <ic> accepte de négocier et comme la propriété spécifiée décrit une propriété extrafonctionnelle qui représente un indicateur de fonctionnement du système, <ic> propose, suite à la demande de relâchement de l'initiateur, le retrait de la clause avec l'alternative RELEASE. La contrainte sur le taux d'affectation est retirée et la négociation par relâchement se termine avec succès. Dans un autre cas de Fig., <ic> peut aussi proposer, comme relâchement, une nouvelle clause pour exprimer sa capacité à s'appuyer sur une contrainte moins forte : $\mathcal{A}_{inv,<ic>} := \{\Psi\}$ avec Ψ, décrivant la nouvelle clause :

<ig>.attributes.getGroupedUsersRatio() >= 60

Cette nouvelle clause va remplacer complètement la clause négociée pour spécifier un taux d'affectation moins élevé (60 au lieu de 80).

Politique par effort. Dans la politique par effort, le gestionnaire de contrats de <ic> consulte <ic> ici dans son rôle de garant. Nous partons ici aussi du principe que le composant <ic> va exploiter alors tout d'abord sa responsabilité de garant vis-à-vis de son assemblage pour négocier par effort, et propager la négociation (voir Fig. A.7). Ainsi, après la phase de consultation initiale pour la négociabilité de la clause, <ic> propose de prendre en charge la négociation car la propriété négociée (`groupedUsersRatio`) est implémentée au niveau de son sous-composant <ig> (*étape 1*). Le contexte de la négociation atomique est alors transmis au gestionnaire de contrats de <ic> qui prend en charge la négociation , et consulte <ig> pour des efforts (*étape 2*). Par la suite, comme <ig> ne peut pas proposer de modifications à son niveau mais qu'il possède, au travers de la formule compositionnelle : <ig>.groupedUsersRatio := $\frac{\text{<gmgr>.nbGroupedUsers}}{\text{<umgr>.nbUsers}}$, des éléments pour orienter la propagation de la négociation, <ig> propose alors de propager à son tour la négociation. La négociation est ainsi transmise au gestionnaire de contrats de <ig> (*étape 3*), qui exploite la formule compositionnelle liant la propriété `groupedUsersRatio` de <ig> aux deux attributs `nbUsers` et `nbGroupedUsers` de <umgr> et <gmgr>. Par la suite, <ig> s'adresse alors à ces deux derniers composants pour obtenir des propositions d'efforts visant à revalider la clause négociée (*étape 4*).

En revanche, comme chacune de ces deux propriétés caractérise le fonctionnement du composant associé en décrivant le nombre d'utilisateurs enregistrés ainsi que le nombre d'utilisateurs ajoutés dans des groupes, ces propriétés représentent des indicateurs de fonctionnement et ne sont pas négociables. Les composants <umgr> et <gmgr> ne peuvent pas proposer des efforts et signifient avec STOP leur incapacité à négocier. Cette négociation se termine en échec, et le gestionnaire de contrats de <ig> retransmet le contexte de négociation au niveau du gestionnaire de contrats initial de <ic> pour que celui-ci finalise la négociation.

Dans cet exemple, la négociation initialement déclenchée au niveau de <ic> est propagée dans l'intérieur de <ig> lequel exploite, cette-fois ci, les composants <umgr> et <gmgr> à partir de la relation compositionnelle qui lie la réalisation de l'attribut `groupedUsersRatio` de <ig> aux attributs `nbGroupedUsers` de <gmgr>, et `nbUsers` de <umgr>. Toutefois, comme ces propriétés décrivent le fonctionnement de ces deux composants et ne peuvent être négociées, les deux composants ne peuvent pas proposer d'efforts pour revalider la clause. La négociation par effort s'est propagée et s'achève alors par un échec.

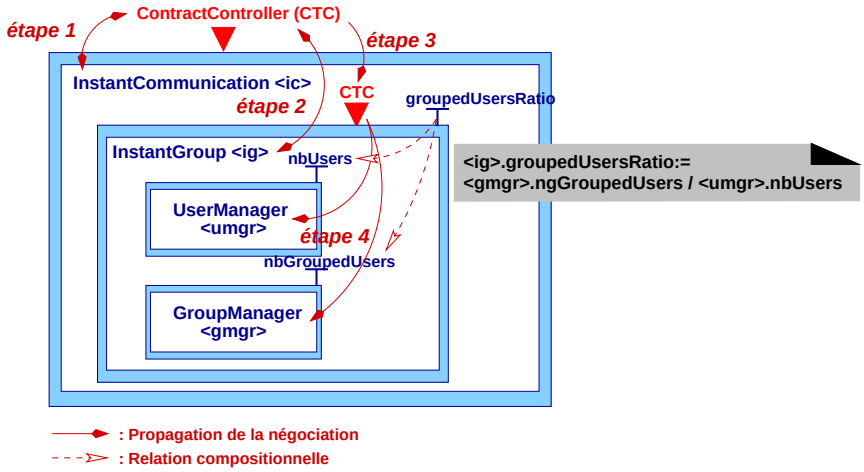


FIGURE A.7 – *Schéma de propagation pour la propriété groupedUsersRatio*

Clause sur la consommation de bande-passante Cette clause établit une contrainte sur le seuil maximal de bande-passante consommée tout le long de l'exécution de toutes les méthodes de toutes les interfaces de tous les sous-composants de <ic>. Elle explicite une contrainte sur un seuil de consommation de bande passante de 30 ko/sec. Il s'agit d'une contrainte de consommation de ressources nécessaire au fonctionnement du système. Elle est vérifiée à l'exécution puisqu'elle concerne la consommation en ressources du système à l'exécution, et est violée par exemple si, la bande-passante consommée excède les 30 ko/sec. Dans ce cas, une négociation atomique démarre alors avec les mêmes parties que dans les clauses précédentes : le gestionnaire de contrats de <ic> dans le rôle de l'initiateur, et <ic> lui-même dans le rôle de bénéficiaire et garant.

Politique par relâchement. Dans la politique par relâchement, le gestionnaire de contrats de <ic> demande à ce-dernier la négociabilité de la clause. Nous supposons que le composant <ic> accepte et propose, suite à la demande de relâchement de l'initiateur, le retrait de la clause avec l'alternative RELEASE. Comme il est l'unique bénéficiaire, la clause est retirée et la négociation par relâchement se termine avec succès. En revanche, plus aucune vérification sur la consommation de la bande-passante n'est effectuée.

Politique par effort. Dans la politique par effort, le gestionnaire de contrats de <ic> consulte <ic> dans son rôle de garant (voir Fig. A.8). A la consultation initiale entre l'initiateur et le garant <ic>, ce-dernier peut alors proposer de prendre en charge la négociation, car la propriété négociée est réalisée au niveau de son sous-composant <bm> (*étape 1*). La négociation est alors transmise au gestionnaire de contrats de <ic> qui prend en charge la négociation, et consulte <bm> pour des efforts (*étape 2*). Comme <bm> ne peut pas proposer de modifications à son niveau mais qu'il possède des éléments pour orienter la propagation de la négociation, il peut alors proposer de propager à son tour la négociation. En effet, d'après la formule compositionnelle : <bm>.BdWLevel() := <vdsm'>.BdWLevel() + <msm'>.getBdWLevel(), la propriété BdWLevel implémentée au niveau du composant <bm> se trouve liée à celle au niveau des sous-composants <vdsm'> et <msm'>, et ces derniers peuvent alors être consultés. La négociation est ainsi transmise au gestionnaire de contrats de <bm> pour que ce dernier prenne en charge la propagation de la négociation dans son intérieur (*étape 3*) et consulte ses deux sous-composants <vdsm'> et <msm'> (*étape 4*). Toutefois, comme les composants <vdsm'> et <msm'> ne représentent ici que de simples sondes de ressources associées aux composants métiers <vdsm> et <msm> et sans

capacités de négociation, le gestionnaire de contrats de <bm> exploite tout d'abord la relation de partage entre les composants <vdsm> et <vdsm′ >, et les composants <msm> et <msm′ > (*étape 5*), puis la relation de contenance pour obtenir les références aux composants métiers englobants <vs> et <ms> (*étape 6*. Contrairement aux composants de sondes précédents, ces derniers sont en effet au premier plan pour négocier puisqu'ils participent directement à la consommation de bande-passante mesurée par leur sonde, et ont potentiellement des capacités pour agir au niveau applicatif. Ainsi, le gestionnaire de contrats de <bm> pilote la négociation et formule des demandes d'efforts à <vs> et <ms> (*étape 7*), , qui en fonction de leur capacités de négociation peuvent proposer des efforts consistant à modifier certains de leur paramètres de fonctionnement (réduction du *frameRate* ou du nombre de clients pour le service vidéo) pour revalider la clause.

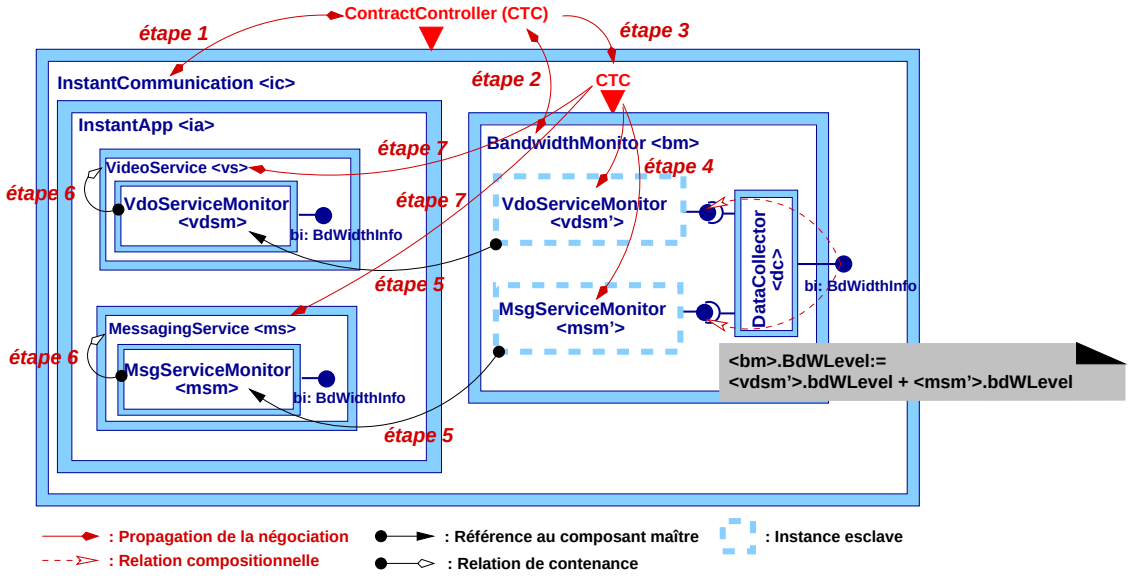


FIGURE *A.8 – Schéma de propagation pour la propriété BdWLevel*

A.2.5 Contrat sur l'interface UserManagement

Clause sur la validité du nom d'utilisateur La clause du contrat d'interface décrite à la Fig. A.9 est vérifiée à l'exécution du système puisqu'elle concerne la valeur effective du paramètre `username` à l'appel de la méthode `createUser`. Si le nom d'utilisateur se réduit à une chaîne vide, aux espaces près, la clause est violée et une négociation atomique démarre. La négociation porte alors sur cette clause, et les composants impliqués sont le gestionnaire de contrats de <ic> dans le rôle de l'initiateur, le composant bénéficiaire <ig> et le composant garant <if>. Diverses formes de négociation peuvent alors se dérouler en fonction des politiques de négociation.

Politique par relâchement. Avec une politique par relâchement, les parties de la négociation sont le gestionnaire de contrats de <ic> dans le rôle de l'initiateur, et <ig> en tant que bénéficiaire, car il a besoin de s'appuyer sur cette contrainte pour pouvoir réaliser le service `createUser`. Comme cette clause porte sur un aspect fonctionnel (un paramètre d'appel de méthode), sa négociation est plutôt limitée. De plus, celle-ci dépend notamment de l'étape du développement logiciel au cours de laquelle la violation a lieu (tests d'intégration, validation, production). Il y a en effet plus de degrés de liberté, et donc de possibilités de négociation, sur de telles clauses lors des phases de tests, que lors des phases de validation et de production.

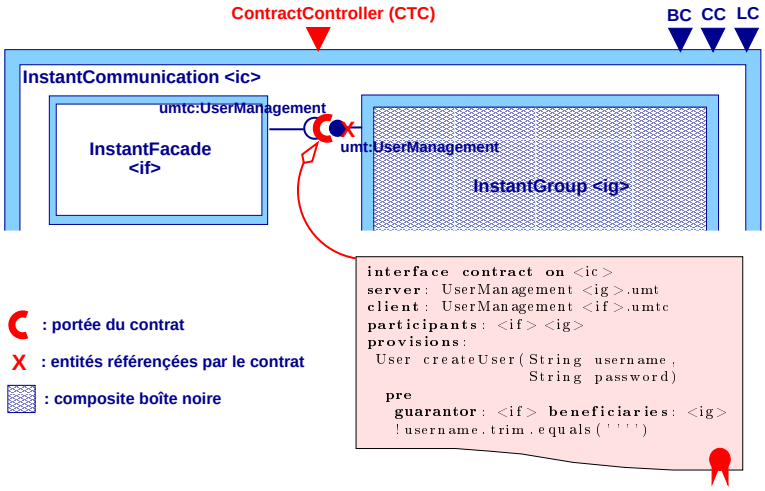


FIGURE A.9 – *Contrat d'interface sur UserManagement*

Néanmoins, pour l'illustrer jusqu'au bout, le processus de négociation est décrit par la suite. Pour démarrer la négociation, le gestionnaire de contrats de `<ic>` demande au seul bénéficiaire `<ig>`, la négociabilité de cette clause. En fonction des capacités de négociation de `<ig>`, soit celui-ci accepte de négocier, soit il refuse et dans ce cas, comme il est l'unique bénéficiaire, la clause est non négociable, et la négociation s'arrête pour cette politique. Dans le cas où la négociation se poursuit, l'initiateur demande alors à `<ig>` de faire des relâchements. Par la suite, compte tenu de la propriété spécifiée, nous supposons que le bénéficiaire ne peut que proposer soit l'arrêt de la négociation tout en conservant la clause, avec l'alternative STOP, soit le retrait complet de la clause en échec, avec l'alternative RELEASE. Le choix entre ces deux alternatives se fait selon le degré d'importance qui est accordé à cette vérification qui est fonction notamment de la phase à laquelle intervient la violation et des contextes de déploiement. Dans les deux cas, comme `<ig>` est le seul bénéficiaire, le gestionnaire de contrats valide la demande et effectue soit l'arrêt de la négociation, soit le retrait la clause. Dans le premier cas, la négociation échoue alors pour cette politique, et c'est au gestionnaire de contrats d'envisager d'autres mécanismes de négociation ou de rattrapage. Dans le second cas, la négociation s'achève avec réussite, et comme la clause est retirée, il n'y a plus de vérification à ce niveau et c'est à `ig` ou au système de supporter ce retrait et d'assurer un fonctionnement cohérent en conséquence.

Politique par effort. Avec la politique par effort, les parties négociantes sont maintenant le gestionnaire de contrats de `<ic>` avec le composant garant `<if>` qui s'engage à satisfaire la clause, car client du service `createUser()` et responsable de son appel. Ainsi, le gestionnaire de contrats peut demander au garant `<if>`, la négociabilité de cette clause. Par la suite, comme cette clause porte précisément sur l'invocation d'une méthode par le composant garant, et qu'il n'y a ici pas d'autres possibilités d'efforts qui consisteraient à réaliser une action de négociation ou une propagation, le composant garant ne peut alors que proposer le refus de la négociation. Du point de vue de `<if>`, cette clause n'a pas à être négociée, elle doit être satisfaite en fonctionnement normal, et dans le cas inverse, il s'agit vraisemblablement d'une erreur d'utilisation du service dans le fonctionnement interne de `<if>`, et cette violation doit être détectée. La négociation par effort s'achève ici par un échec.

A.3 Détails des architectures

Cette section décrit les architectures internes des composants `InstantGroup`, `InstantApp` et `BandwidthMonitor`.

A.3.1 Architecture du composant InstantGroup

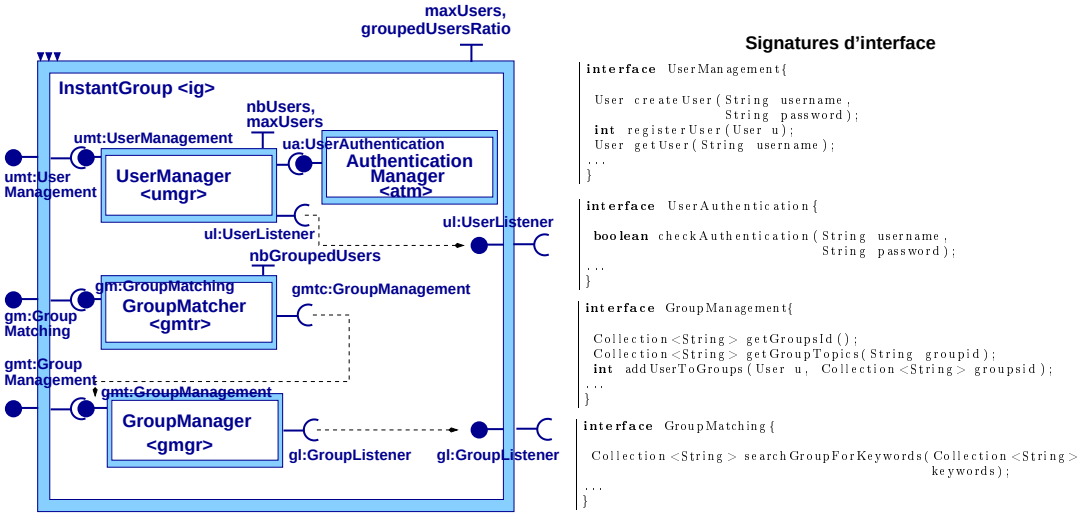


FIGURE *A.10 – Architecture interne du composant InstantGroup*

L'intérieur du composite `InstantGroup` est formé de quatre sous-composants avec les principales fonctionnalités fournies et requises suivantes (cf. Fig. A.10) :

- `UserManager` prend en charge la gestion des utilisateurs (création, enregistrement, etc.). Au travers de son interface fournie `umt :UserManagement`, il permet notamment de créer de nouveaux utilisateurs (`createUser()`), d'enregistrer des utilisateurs existants dans le système (`registerUser()`), de récupérer des utilisateurs crées (`getUser()`). Ce composant, au travers de son interface requise `uac :UserAuthentication`, utilise des services d'authentification des utilisateurs. Il émet aussi des événements liés aux utilisateurs par son interface requise `un :UserListener`.
- `AuthenticationManager` apporte le service technique qui permet d'authentifier des utilisateurs. La méthode `checkAuthentication()` de l'interface `ua :UserAuthentication` permet de vérifier la validité d'un utilisateur à partir d'un couple (login,mot de passe).
- `GroupManager` prend en charge la gestion des groupes (création, abonnement d'utilisateurs, etc.). Son interface fournie `gmt :GroupManagement` permet de récupérer les identifiants de tous les groupes (`getGroupsId()`), de récupérer les thèmes d'un groupe donné (`getGroupTopics()`), et d'ajouter un utilisateur dans un groupe (`addUserToGroups()`). De plus, il émet des événements liés aux groupes par son interface requise `gn :GroupListener`.
- et enfin, `GroupMatcher` permet de faire le lien entre les mots-clés fournis par les utilisateurs et les groupes qui y correspondent. L'interface fournie `gm :GroupMatching` comporte notamment la méthode `searchGroupsForKeywords()` qui effectue l'appariemment entre les centres d'intérêts fournis par les utilisateurs (*keywords*) et les thèmes des groupes (*topics*). À partir des mots-clés fournis par les utilisateurs, il détermine et rend les identifiants des groupes auxquels

il faudrait les ajouter. La dépendance aux services liés aux groupes est exprimée par l'interface requise `gmtc :GroupManagement`, connectée à l'interface fournie `gmt :GroupManagement` de `GroupManager`.

A.3.2 Architecture du composant InstantApp

L'architecture du composite `InstantApp` est, quant à lui, formé de quatre sous-composants (cf. Fig. A.11). Il encapsule un composant de contrôle des services `ServiceFacade` et les sous-composants qui permettent d'établir et de gérer les différents services de communication partagés par les utilisateurs : `VideoService` et `MessagingService` proposent les services de diffusion vidéo et de messagerie instantanée. Ce sont ces services qui sont adaptés aux différents thèmes des groupes et aux centres d'intérêts des utilisateurs appariés. Ces composants exposent les principales fonctionnalités fournies et requises suivantes :

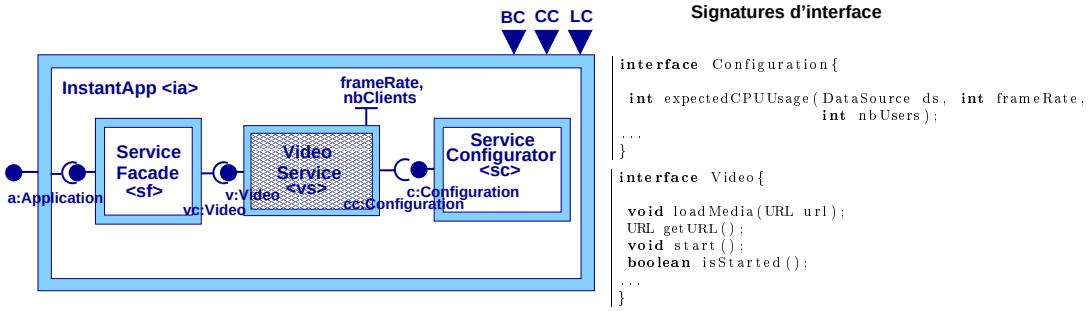


FIGURE *A.11 – Architecture interne du composant InstantApp*

- `ServiceFacade` permet de piloter l'ensemble des services disponibles. L'interface `a :Application` permet, notamment, d'invoquer les services liées à la diffusion vidéo (`start()`) et à la messagerie instantanée (`sendMessage(), getNextMessage()`).
- `VideoService` offre les services liés à la diffusion de vidéo. Les méthodes `loadMedia()` et `start()` de l'interface `v :Video` permettent respectivement de charger et de débuter la diffusion vidéo.
- `MessagingService` offre aux différents utilisateurs d'un groupe, les services classiques liés à la messagerie instantanée (envois et réceptions de message, etc.) par l'interface `m :Messaging`.
- `ServiceConfigurator` fournit divers services pour configurer et optimiser les services précédents. Par exemple, la méthode `expectedCPUUsage` s'adresse au service de diffusion vidéo et permet d'estimer l'utilisation du CPU induite par la diffusion vidéo en tenant compte de certains paramètres comme le *frameRate* (`frameRate`) et le nombre de clients du service (`nbClients`). Les services fournis par ce composant sont utilisés par `VideoService` et `MessagingService`. Enfin, comme les composants `VideoService` et `MessagingService` proposent des services liés aux utilisateurs et aux groupes (connexion d'utilisateur, mise à jour des groupes, etc.), ils sont aussi notifiés de différents événements qui les concernent. Cette dépendance est externalisée au niveau du composant `InstantApp` par les interfaces fournies `unc :UserListener` et `gnc :GroupListener`.

A.3.3 Architecture du composant BandwidthMonitor

L'architecture du composite est formée de trois sous-composants (cf. Fig. A.12).

Le composant `BandwidthMonitor` contient trois composants `VdoServiceMonitor` et `MsgServiceMonitor` qui fournissent les consommations de bande passante associées aux composants métiers `VideoService` et `MessagingService` de `InstantApp`, ainsi que `DataCollector` qui a en charge la collecte de toutes ces informations. De plus, comme les composants `VdoServiceMonitor` et `MsgServiceMonitor` mesurent la consommation de bande-passante de `VideoService` et `MessagingService` respectivement, ils sont partagés et sont contenus à la fois dans `VideoService` et `<bm>`, pour `VdoServiceMonitor`, et, dans `MessagingService` et `<bm>`, pour `MsgServiceMonitor`. Plus précisément, le partage des composants `VdoServiceMonitor` et `MsgServiceMonitor`, applique le patron maître-esclave de sorte qu'il n'existe qu'une instance du composant `VdoServiceMonitor`, que nous nommons `<vdsm>`, contenu dans `VideoService` et avec `<vdsm'>` composant esclave de `VdoServiceMonitor` contenu dans `<bm>` qui référence `<vdsm>`. Le même schéma s'applique pour le composant maître `<msm>` contenu dans `MessagingService`, avec `<msm'>` contenu dans `<bm>` et ayant une référence sur l'instance maître `<msm>`. Ainsi, par `<vdsm'>` et `<msm'>`, les consommations des services de vidéo et de messagerie collectées sont accessibles, collectées par `DataCollector`, et exposées au niveau de l'interface `bi :BdWidthInfo` de `BandwidthMonitor`.

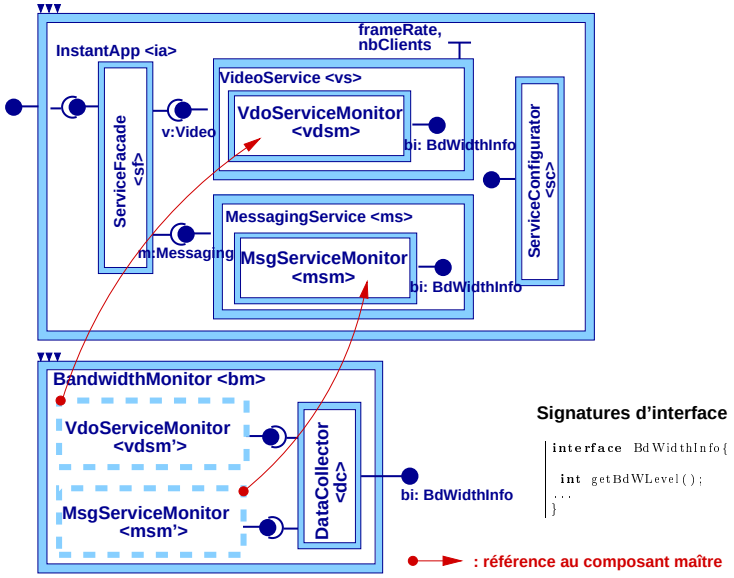


FIGURE *A.12 – Architecture interne du composant BandwidthMonitor*

B

Compléments sur le système ConFract

Dans cette annexe, nous décrivons plus en détails les principales caractéristiques de la première version du système de contractualisation *ConFract* [Collet *et al.* 2005], précédemment conçu dans l'équipe RAINBOW. Les objectifs et les principes généraux de *ConFract* sont tout d'abord présentés. Puis, les principaux éléments des contrats (types de contrats, contenu, responsabilités et cycle de vie) sont décrits.

B.1 Objectifs et principes

Le système *ConFract* est proposé avec pour objectif principal d'appliquer l'approche contractuelle aux composants logiciels hiérarchiques. Ainsi, en exploitant la richesse et les qualités de généralité du modèle de composant *Fractal*, *ConFract* se présente comme un système pour organiser sous forme de contrats la spécification et la vérification de diverses propriétés sur les composants logiciels *Fractal*. Compte tenu de ces objectifs, les principes généraux du système *ConFract* sont les suivants :

- les *spécifications*, qui sont exprimées dans un formalisme, sont clairement distinguées des *contrats*, qui sont réifiés et dynamiquement construits à partir des spécifications. Les contrats capturent ainsi diverse propriétés fonctionnelles et extrafonctionnelles attendues des systèmes, et suivent leur cycle de vie.
- les contrats sont dynamiquement construits en fonction des événements de configuration des composants, et sont mis à jour lors des reconfigurations dynamiques. Ils représentent ainsi une source d'informations à jour qui peut être exploitée à des fins de surveillance, de vérification, etc.
- les contrats contiennent très précisément des responsabilités aux composants impliqués. Ces responsabilités sont attribuées par le système *ConFract*, et constituent un élément majeur pour organiser avec précision les violations de contrats ;
- divers types de contrats sont définis par *ConFract* afin de prendre en compte les spécificités du modèle général *Fractal* (hiérarchie et visibilité des composants). De plus, l'intégration de *ConFract* au modèle *Fractal* respecte au mieux les principes de ce dernier (séparation des préoccupations, possibilités de re-configurations dynamiques).

Dans *ConFract*, les contrats sont construits, à partir de spécifications, lors des assemblages de composants. Ces contrats sont alors des objets de première classe pendant les phases de configuration et

d'exécution des composants. *ConFract* a été conçu pour séparer clairement les mécanismes contractuels de l'expression des spécifications. La première version de ce système se base sur un langage d'assertions exécutables, nommé *CCL-J* (*Component Constraint Language for Java*). Ce langage, partiellement inspiré d'*OCL* [OMG 1997], est dédié à *Java* et *Fractal*. *CCL-J* supporte les catégories de spécification classiques qui correspondent à des périodes de validité dans le processus d'exécution – préconditions (*pre*) à l'entrée des méthodes, postconditions (*post*) à la sortie des méthodes, invariants de configuration (*inv*) durant toute la période d'exécution et *rely* tout le long de l'exécution de méthodes – et dont la portée est associée aux interfaces (construction *context*), mais aussi aux composants (construction *on*).

B.2 Types de contrat

Le système *ConFract* introduit différentes formes de contrat pour tenir compte des spécificités du modèle *Fractal*. Ainsi, trois types de contrats sont distingués :

- un *contrat d'interface* contractualise une relation client-fournisseur. Il est établi sur chaque connexion entre une interface requise et une interface fournie. Il regroupe les spécifications des deux interfaces. Ces spécifications ne peuvent référencer que les méthodes de l'interface, établissant un contrat comme on le définit dans les systèmes à objets. Par exemple, la figure B.1 montre un contrat d'interface établi au niveau de la connexion entre l'interface requise `a` du composant `A` et l'interface fournie `b` du composant `B`. Ce contrat peut référencer uniquement tout élément qui concerne le couple d'interface formé de `a` et `b`.

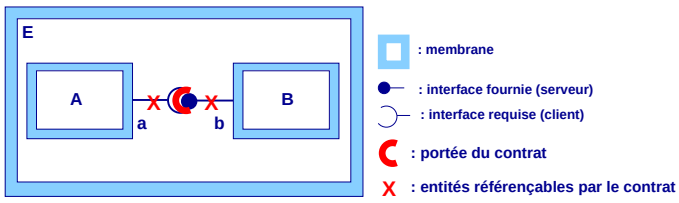


FIGURE *B.1 – Portée et entités référençables dans un contrat d'interface*

ConFract introduit aussi des *contrats de composition*, qui sont associés à la membrane des composants et qui regroupent de manière spécifique les spécifications définies sur les membranes de composants. Selon la portée de visibilité qu'il est possible d'avoir de chaque côté de la membrane – de la membrane vers l'extérieur du composant ou de la membrane vers l'intérieur du composant – deux formes de contrat de composition sont définis :

- un *contrat de composition externe* contractualise l'usage d'un composant vis-à-vis de son environnement externe. Il est construit à partir des spécifications qui ne référencent que des interfaces visibles à l'extérieur de la membrane et placées sur celle-ci. En particulier, une caractéristique majeure du contrat de composition externe réside dans le fait que celui-ci peut référencer plusieurs interfaces sur un même composant. Par exemple, la figure B.2 montre un contrat de composition externe établi au niveau de la membrane du composant `B` et qui porte vers l'extérieur de ce composant. Ce contrat peut référencer uniquement tout élément qui concerne les interfaces externes, `b1`, `b2`, `b3`, de `B`.
- un *contrat de composition interne* contractualise l'assemblage et le comportement d'un composant vis-à-vis de son environnement interne. Il ne contient, quant à lui, que les spécifications qui référencent des interfaces dans le contenu du composite. Les interfaces référencées peuvent ainsi être soit des interfaces internes du composite, soit des interfaces externes de ses sous-composants. Par exemple, la figure B.3 montre un contrat de composition interne établi au niveau de la mem-

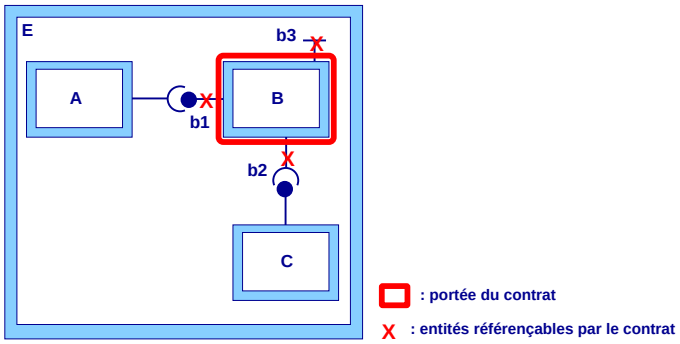


FIGURE *B.2 – Portée et entités référençables dans le contrat de composition externe*

brane du composant `E` et qui porte vers l'intérieur de ce composant. Ce contrat peut référencer uniquement tout élément qui concerne les interfaces internes de `E` ainsi que les interfaces externes des sous-composants, `A` et `B`, de `E`.

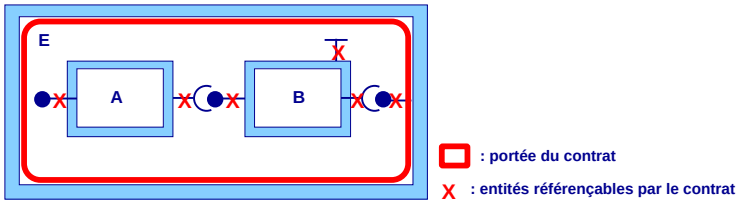


FIGURE *B.3 – Portée et entités référençables dans le contrat de composition interne*

Pour résumer, le contrat de composition externe exprime l'usage d'un composant et des règles externes liées à son utilisation, alors que le contrat de composition interne d'un composite exprime son assemblage et des règles liées à son implémentation.

B.3 Contenu des contrats et responsabilités

Les contrats construits par le système *ConFract* regroupent les spécifications suivant leur catégorie et chaque clause de spécification devient une *clause* du contrat. Au moment de construire les contrats, *ConFract* exploite les capacités réflexive des composants *Fractal* et inclut dans chaque clause de contrat, la spécification (par exemple une assertion *CCL-J*), le contexte d'interception (indiquant le contexte spatio-temporel dans lequel la clause doit être satisfaite) ainsi que les références au contexte (composants, interfaces, etc.) nécessaires à la vérification de la clause. Les règles de vérification des contrats sont détaillées dans la suite à la sous-section B.4.

Ces contrats contiennent aussi les références aux composants participants à l'évaluation de chacune des clauses. Un métamodèle établit en effet clairement pour chaque type de spécification (par exemple les préconditions d'une méthode dans une spécification externe de composant), les responsabilités attribués aux composants impliqués. Ces responsabilités deviennent ainsi effectives au moment de la construction du contrat. Comme les contrats attribuent très précisément des rôles aux composants qui y participent, ils sont construits progressivement au fur et à mesure de l'ajout des composants dans le système. Les règles de construction des contrats sont données dans la sous-section B.4. La suite détaille les responsabilités attribuées par *ConFract*. Ces responsabilités peuvent ainsi être :

– des *garants*, représentant les composants qui s'efforçent de rendre la clause vraie. Ils doivent être

prévenus en cas d'échec de vérification de la clause et ils ont la capacité d'agir pour traiter ces échecs,
- des *bénéficiaires*, représentant les composants qui veulent pouvoir compter sur la véracité de la clause et qui peuvent être prévenus en cas de changement d'état de celle-ci (invalidation ou rétablissement),
- et d'éventuels *contributeurs*, qui sont des composants qui participent à la véracité de la clause et qui sont donc nécessaires à l'évaluation de cette dernière.

B.3.1 Responsabilités du contrat d'interface

Dans le cas d'un contrat d'interface, les responsabilités sont directement celles d'une relation client/-fournisseur, exactement comme dans les contrats objet.

Ainsi, pour un contrat d'interface établi au niveau d'une connexion entre l'interface client `a` d'un composant client `A`, et l'interface serveur `b` d'un composant fournisseur `B` (cf. Fig. B.1), les responsabilités, pour des pré et postconditions sur des méthodes de l'interface serveur `a` et client `b` de `B`, sont données par la table suivante :

Type d'interface	Construction	Garant	Bénéficiaire
serveur (`b`)	pre	`A`	`B`
serveur (`b`)	post	`B`	`A`
client (`a`)	pre	`B`	`A`
client (`a`)	post	`A`	`B`

TABLE *B.1 – Responsabilités associées au contrat d'interface pour des pré et postconditions sur une interface serveur et client*

En effet, les pré et postconditions posées au niveau de méthodes d'interfaces serveur expriment des contraintes sur l'exécution des services. Ainsi, une précondition sur une méthode de l'interface `b` exprime une contrainte devant être satisfaite pour que le début de l'exécution par le composant `B` se fasse avec succès : `B` est donc bénéficiaire de la précondition satisfaite car il bénéficie de la possibilité d'exécuter le service, et `A` en est le garant. Réciproquement, une postcondition sur une méthode de cette même interface exprime une contrainte qui doit être satisfaite par le composant `B` à la fin de l'exécution de la méthode : `B` est donc le garant car il s'engage à satisfaire cette contrainte à la fin de l'exécution de la méthode qu'il (ou son assemblage) implémente, et `A` en est le bénéficiaire.

À l'inverse, pour des méthodes d'interfaces client, les pré et postconditions expriment, cette fois-ci, des contraintes sur l'invocation des services ; les responsabilités entre `A` et `B` sont alors inversées. Une précondition sur une méthode de l'interface `a` exprime une contrainte devant être satisfaite pour que le début de l'invocation de la méthode par le composant `A` se fasse avec succès : `A` est donc le bénéficiaire de la précondition satisfaite car il bénéficie de la possibilité d'invoquer le service, et `B` en est le garant. Réciproquement, une postcondition sur une méthode de cette même interface exprime, quant à elle, une contrainte qui doit être satisfaite par le composant `A` à la fin de l'invocation de la méthode : `A` est donc le garant car il s'engage à satisfaire la contrainte à la fin de l'invocation de la méthode, et `B` en est le bénéficiaire.

B.3.2 Responsabilités du contrat de composition externe

Pour un contrat de composition externe, dans le cas de la configuration de la figure B.2, les responsabilités sont données par la table suivante :

Type d'interface	Construction	Garant	Bénéficiaires
serveur (b1)	pre	E	B
serveur (b1)	post	B	E, A
client (b2)	pre	B	E,C
client (b2)	post	E	C

TABLE *B.2 – Responsabilités associées au contrat de composition externe pour des pré et postconditions sur une interface serveur et client*

Par exemple, pour une précondition sur une méthode serveur (interface b1) du composant B, le garant est le composant englobant E, et le bénéficiaire est le composant B. Le garant est E, et non pas le composant A, car E est justement l'englobant qui contient B, alors que A, est simple utilisateur de l'interface b1 (A serait garant, par exemple, du contrat d'interface sur la connexion). En effet, comme le contrat de composition externe décrit ici l'usage du composant B, c'est à E, l'englobant, que revient la responsabilité d'assurer la bonne utilisation générale de ses sous-composants et donc l'usage correct de B vis-à-vis des autres sous-composants.

Pour une postcondition sur une méthode définie sur l'interface b1 de B, le garant est le composant B lui-même car il implémente le service fourni. Les bénéficiaires sont le composant utilisateur du service A ainsi que l'englobant E, car A est connecté et client du service et E contient B.

Pour une méthode définie sur l'interface client b2 de B, les responsabilités d'une précondition sont B en tant que garant de de l'invocation de la méthode, E et C, en tant que bénéficiaires, respectivement de l'usage de B, et de l'invocation de la méthode par B. Les responsabilités d'une postcondition sont E en tant que garant de l'usage de B, et C le bénéficiaire car fournisseur du service invoqué par B.

Pour des postconditions et sur b1 et des préconditions sur b2, les rôles de bénéficiaires sont affinés pour prendre en compte la différence de visibilité entre les deux composants bénéficiaires. Ainsi, E est le bénéficiaire principal dans les deux cas car il est l'englobant de B et voit potentiellement tout son contenu alors que A (resp. C) ne voit que l'interface b1 (resp. b2) de B.

B.3.3 Responsabilités du contrat de composition interne

Enfin, les responsabilités associées au contrat de composition interne sont plus simples. Le composant englobant qui est porteur du contrat (le composant E dans la configuration de la figure B.3) est à la fois le seul garant et le seul bénéficiaire. Ce contrat régit en effet les contraintes d'architecture et de comportement du composite vis-à-vis de ses sous-composants, le composite est alors seul responsable de son *implémentation* ; il garantit son assemblage et bénéficie de la justesse de celle-ci. De plus, comme il est englobant, il a la vision sur tout ce qui concerne à la fois son intérieur et l'extérieur de ses sous-composants.

Enfin, il est à noter que quelque soit le type de contrat considéré, les composants responsables d'un contrat sont, soit au même niveau de la hiérarchie (contrat d'interface et contrat de composition interne), soit liés dans une relation composite/sous-composants (contrat de composition externe). De plus, pour ces trois types de contrats définis par *ConFract*, le garant est unique.

B.4 Cycle de vie des contrats

B.4.1 Gestion des contrats

Pour gérer les contrats dans *ConFract*, des contrôleurs de contrats (`CTC`) sont introduits et placés sur la membrane de chaque composant composite. Chaque gestionnaire de contrats prend en charge le cycle de vie et l'évaluation *(i)* du contrat de composition interne du composite sur lequel il est placé, *(ii)* du contrat de composition externe de chacun des sous-composants et *(iii)* du contrat d'interface de chaque connexion dans son contenu. Le gestionnaire de contrats utilise le mécanisme de mixin [16] de l'implémentation en Java de *Fractal* pour effectuer des intercessions sur les contrôleurs de cycle de vie, de connexion et de contenu. Il construit et met à jour les contrats en fonction des événements d'assemblage qui surviennent sur les composants (insertion d'un sous-composant, connexion de deux interfaces, etc.).

B.4.2 Construction et mise à jour des contrats

Lorsqu'un composant est introduit dans un assemblage, *ConFract* crée un contrat de composition interne s'il est composite, et un contrat de composition externe s'il a des spécifications liées à plusieurs de ses interfaces externes. Pour chaque spécification liée à des contrats de composition, un patron de clause (*provision template*) est crée et attaché au contrat de composition. Chaque patron de clause est alors en attente de tous ses participants (les participants effectifs de la clause) pour *se fermer* et devenir une clause. Ainsi, lorsqu'un nouveau sous-composant est ajouté à une composition de composants, tous les patrons qui participent aux contrats de composition concernés, ont leurs responsabilités qui sont complétées. Dès que tous les contributeurs d'un patron sont connus, celui est fermé et devient une clause. Lorsque tous les patrons de clause d'un contrat de composition interne sont fermés, le contrat est lui aussi *fermé*, car toutes les responsabilités sont identifiées pour chacune de ses clauses. De fait, les contrats sont alors armés et le composant peut finallement être démarré. Pour un contrat d'interface entre un client et un serveur, le cycle de vie est plus simple car il n'y a seulement que deux participants aux contrats, liés par une relation de type client/fournisseur. Ce contrat est alors créé lors de la connexion des interfaces des composants client et serveur, et il est ensuite immédiatement fermé, puisqu'alors toutes les responsabilités sont connues.

Lorsque des reconfigurations dynamiques sur l'architecture des composants s'opèrent (ajout, remplacement, suppression de composants, etc.), les contrats impactés (pour lequels au moins un participant manque ou est modifié) sont mis à jour. Le retrait de composants entraîne alors la réouverture des contrats auxquels ces derniers participent, et la clause concernée redevient à l'état de patron de clause puisqu'au moins un participant manque. Lorsque de nouveaux composants sont ajoutés au système, les contrats et les patrons de clause évoluent progressivement vers leur fermeture selon le mécanisme décrit précédemment. Ces composants se voient alors attribuées par *ConFract*, des responsabilités soit nouvelles, soit récupérées de celles des composants qu'ils remplacent. Ainsi, les contrats suivent les reconfigurations dynamiques d'architecture des composants, et confèrent au système *ConFract* une certaine robustesse vis-à-vis de ces reconfigurations. Ce dernier peut alors à nouveau poursuivre ses activités de vérification des nouveaux contrats à jour, et de détection précise de leurs violations.

B.4.3 Vérification des contrats

Bien que d'autres formes de vérification soient potentiellement intégrables, le système *ConFract* effectue actuellement les vérifications des clauses des contrats en les évaluant. Ces évaluations se font de la façon suivante. Une fois les contrats armés, les clauses des contrats de configuration

16. Un mixin est une classe dont la super-classe est définie de manière abstraite par l'ensemble des primitives uniquement nécessaires ; les différents mixins d'un gestionnaire sont ainsi fusionnés en une seule classe au chargement.

qui définissent des invariants sur les composants sont vérifiées à la fin de la phase de configuration. Dans le cas du langage *CCL-J*, toutes les autres clauses sont vérifiées à l'exécution des composants car les assertions peuvent être paramétrées par des éléments fonctionnels d'exécution (méthodes, valeurs effectives des paramètres). Lorsqu'une méthode est appelée sur une interface *Fractal*, les préconditions du contrat d'interface sont d'abord évaluées, puis celles du contrat de composition externe du composant recevant l'appel, et enfin celles du contrat de composition interne. Des vérifications, dans l'ordre inverse, sont effectuées au retour de la méthode pour les postconditions et les invariants. Dans les cas où la vérification d'une clause n'est pas satisfaite, la politique par défaut des contrôleurs de contrat consiste à lancer une exception qui décrit tout le contexte de la violation.

Table des figures

Liste des tableaux

Liste des listings